UNIVERSITY OF TROMSØ UIT
FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

A Context-Aware Mobile Bus Application

Christer A. Hansen

INF-3990
Master's Thesis in Computer Science
May, 2010

Abstract

Accessing route information should be easy. Today, most collective transport companies distribute timetables online as electronic documents and in paper format. These solutions are outdated and cumbersome to use. However, systems have been built to make the task of finding route information easy, and to replace these formats. Most of these systems, still, have limitations. They rely on users knowing the name of the bus stops, and the destination is left out when finding information. In this thesis we present a system that is able to find travel alternatives based on two parameters, the user's current location and the destination. Successful tests and experiments have proved that our system can be useful for people that does not know the following; the name of the bus stops, where the bus stops are located, what route to use, and where to get of the bus. We also suggest an architecture where our system is integrated as part of a *personal cloud* [2].

Keywords: PIA, personal information, assets, iPhone, bus application, web-service, REST, Route Information Server, personal cloud, GPS, context-aware.

Acknowledgments

When I moved to Tromsø, after finishing my Bachelor's Degree at HiNe, I had to use collective transport on a daily basis. The frustration of not being able to receive route information on my iPhone annoyed me. So I started thinking what features a bus application should have had. Before finishing the first year of my Master, I suggested the problem to Anders Andersen, my supervisor. He accepted the proposal, and here we are.

I would like to start thanking Anders Andersen for accepting my thesis and for providing valuable ideas, support, and motivation throughout this project. I would also like to thank Anders Engan and Tom Martin Norvang, my fellow master students, for the relevant discussions we have had concerning the PIA project. I would also like to thank Jan Fuglesteg for the administrative support he has given me during my 2-year stay at the University.

A special thanks goes to the guys at mPower AS, Wiggo Finnset and Jan Grav, for providing me with ideas and data sets. I appreciate your interest in my project, and that we are able to cooperate outside the context of my thesis.

The biggest thank you goes out to my girlfriend Veronica for dragging me to Tromsø. This would never have happened if it weren't for you.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

In large cities collective transport plays an important role in the transport strategy. Many cities have to transport people using metros and busses because the lack of space (roads, parking slots, etc.). A bus containing 50 passengers can reduce the total number of cars driving around by 40. With less cars we get smaller amounts of emission and more space. Still, if people are going to choose collective transport instead of driving their own cars, good collective solutions should be made available. Collective transport companies should be flexible, and offer a variety of routes and a large array of departures. It is also important how the companies choose to distribute their timetables.

Timetables are provided by bus companies for users to find out when the bus departs from a given bus stop. The timetables are in most cases provided in two different forms, either as PDF documents distributed online, or as paper timetables made available on the bus or at the bus stops. These formats have two things in common, they are hard to navigate, and time consuming to use. Each timetable often only contain one single route, which means that you need several timetables if you are using several routes. This makes it even harder to keep track of the route information.

Web-based systems have been built to offer users other methods for accessing route information [1] [2]. These systems let the user define his own favorite set of bus stops. By doing this, they require the user to know what the bus stops are named and where they are located. These systems also lack the support for *context-awareness*.

Route information should be made available for devices that are ubiquitous, such as mobile phones. Today, most people own mobile phones that are capable of browsing the Internet, taking pictures, and even knowing their current location. They are also equipped with high resolution screens, which make them able to present almost the same amount of route information as a printed timetable. The advantage of deploying

[1] Neste Avgang webpage: http://www.reiskollektivt.no/tfk/main.php

[2] TravelMagic webpage: http://rp.tromskortet.no/scripts/travelmagic/travelmagicwe.dll/p

route information on a mobile phone is the increased accessibility. Another advantage is that you are not dependent on PDF documents or paper printed tables anymore.

In this thesis we will design and implement a mobile context-aware bus application. The application will be simple, since the only input the user have to give is where he is going. From this point the application will suggest multiple travel possibilities for the user, illustrate where the bus stops are located, and notify the user when he is getting close to the destination. The application will receive all the information it needs from a system we call RIS (Route Information Server). The RIS system will be designed and implemented in this thesis along with the mobile application. RIS extracts route– and location information from different resources and combine them into new ones that fulfill the requirements of the mobile application.

The work presented in this thesis is part of a larger project at the University of Tromsø, called PIA (Personal Information and Assets) [2]. The PIA project focuses on a *personal cloud* that integrates personal information and makes it more available, in our case route information. The personal cloud should also make it easy to create new services by combining information that it possesses. In this thesis we will discuss what RIS would require from such a personal cloud if it were to be integrated as part of it.

1.2 Problem definition and goals

Our goal in this thesis will be to design and implement a system that offers route information in a ubiquitous and simple way. A system will be built offering a set of resources required by a context-aware mobile bus application. The bus application will also be implemented for testing and evaluating the system's functionality. The user's context will play an important role when extracting information. The system will offer its functionality through a web-service as a set of resources. The resources will contain data extracted from a variety of location– and route information services. Our objective is also to address the resource requirements of our system, and to find out what functionality the PIA platform must provide if we were to integrate our system in a personal cloud.

1.3 Interpretation

We are going to implement two components in this thesis, a mobile context-aware bus application, and a RIS (Route Information Server).

The RIS receives requests that contain context parameters from the bus application, such as travel interests and coordinates describing the user's location. It will use these parameters to find information for the client. This is done by extracting data from two different resources, a route information resource, and a location resource. When the RIS have extracted the information, it builds the resource the client requested, and includes the extracted data.

The application will send requests to RIS based on user inputs, asking for bus stops close to a destination, bus stops nearby, and departure times. The application will utilize the user's context to make the user experience as simple and seamless as possible. For instance, the application will use the GPS (Global Positioning System) to pinpoint the user's location.

Our goal is also to address the resource requirements of RIS. The system must have access to route information and a location service for being able to build the resources it offers through the web service. We will discuss and suggest how a personal cloud can offer this kind of information.

The work presented in this thesis is proof of concept. We will implement a system based on our ideas of helping users that are unfamiliar in Tromsø. The functionality will be prioritized. We will not focus on issues such as scalability and availability.

1.4 Method

The discipline of computing is divided into three paradigms [5]. These paradigms are *theory, abstraction* and *design.* This thesis will follow a *prototype* paradigm which is similar to the *design* paradigm. We will work iterative and repeat the following workflows until all our requirements are fulfilled:

1. Requirements.
2. Analysis and design.
3. Implementation.
4. Testing and evaluation.

1.5 Outline

The thesis consists of the following chapters:

Chapter 2 - Related work introduces the domain of the PIA project. Related route information systems are also mentioned in this chapter.

Chapter 3 - Requirements states the requirements of the system.

Chapter 4 - Design proposes a design based on the requirements.

Chapter 5 - Implementation describes the implementation of the RIS and the bus application.

Chapter 6 - Evaluation describes the experiments, and evaluates the results.

Chapter 7 - Conclusion draws the conclusion of this thesis and gives a proposal for future work.

Chapter 2

Related work

In this chapter we will discuss related work. Details about the PIA project will be given, with a list of projects and systems that are related to it. We will also take a closer look on some existing route information systems that are similar to the system we are going to design and implement in this thesis.

2.1 PIA

PIA (Personal Information and Assets) is a project in its initial phase here at the University of Tromsø. The motivation behind this project is a strong belief that a new approach to developing distributed applications involving personal information is needed. The main objectives of the PIA project is to: (1) enhance the availability of information important to individuals, communities, and/or organizations, (2) to provide support for enhanced value and usability of information through integration, sharing and context-awareness, and (3) managing information of importance to individuals, communities, and/or organizations in a safe and secure manner.

It is suggested that a support system, based on the idea of a personal cloud, is deployed between the applications and the resources, as illustrated in Figure 2.1. Resources are defined as personal information or assets.

2.1.1 Personal Information

Personal information covers information that is related to a person, such as information (1) controlled or owned by the person, (2) about the person, (3) directed towards the person, (4) provided by the person, (5) and relevant or useful to the person [11]. This includes personal documents, Internet resources of personal interest, codes, passwords, pictures, videos, health data and much more.

Security is an important topic in PIA since personal information is sensitive.

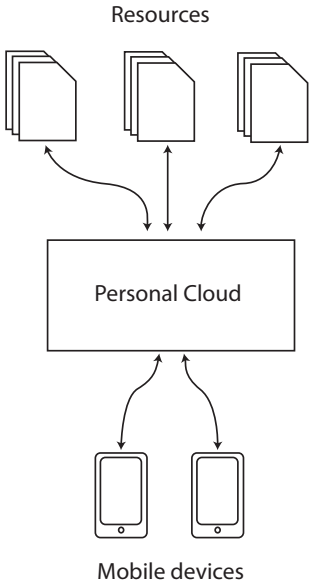

Figure 2.1: The idea of a personal cloud, suggested in PIA.

2.1.2 Assets

Assets are resources that have a value or is of importance to the owner (individual or organization). They are personal information resources with great matter of safety and/or privacy.

2.2 PIM

PIM (Personal Information Management) refers to both the practice and the study of the activities people perform in order to acquire, organize, maintain and retrieve information for everyday use [14]. Computer users manage personal information all the time, either when doing their work or when surfing the web. Our personal information is spread across electronic documents, e-mails, paper documents, instant messages, videos, photographs, etc. This fragmentation makes it hard to find the information we are looking for when we need it. [3] suggests that improving the PIM may save us time, money, energy and attention. They also state that increased PIM will give better productivity in organizations. Increasing the PIM means that people will have access to the right information at the right time, in the right place, in the right form, and in the right quality, to complete the task they are doing.

The goal in the study of PIM is to design new tools that make it easier for users to find the information they are looking for without making it more complex.

2.2.1 SIS

A system called Stuff I've Seen (SIS) was developed for making it easy to re-find information previously accessed [7]. The system was built to index information a person accessed. Wether it was e-mails, web pages, documents, calendar appointments, etc. With SIS all of these sources where integrated into a single index, regardless of what form the information originated. The system was used as a search engine for re-finding previously accessed information. It was tested by more than 230 employees, and they said that finding information was way more easy using SIS than manually browsing or searching.

2.3 Semantic desktops

The core idea behind semantic desktops is to bring semantic web technologies to the desktop, this enables people to use their desktop computers like a personal semantic web, where applications integrate information through ontologies [13].

2.3.1 SEMEX

The SEMEX (SEMantic EXplorer) system introduced in [6] came with the idea of browsing personal information by semantically meaningful associations. The challenge they faced was how they could automatically create associations between data items on a person's computer. The system starts off by extracting data from multiple sources (PDF, Latex, Bibtex, Contacts, Email, Word, etc.). The extractions create instances and classes in a domain model (i.e. a Person class). References to the same real life object is solved by automatically reconciling them. The user can search his personal data in three ways; (1) Keyword search, (2) choosing a class in the domain model and filling in some attributes for the given class, (3) and search based on associations. When a particular object is selected, the user can see all the associated objects.

2.3.2 Haystack

Haystack is a project at the Massachusetts Institute of Technology. The motivation behind this project is to provide high quality information retrieval, and to build a system that adapts to its user, instead of forcing the user to adapt to the limitations of the system. Haystack is implemented as a tool that runs on an individual's own machine. In Haystack they choose to build their data model based on the data objects metadata. The harvesting for the Haystack data model can be done in three different ways; (1) Data driven clients that digest data already in Haystack to produce more data, (2) observers that indexes the information the user is accessing, (3) active human annotation carried out by the user to improve his data organization [1].

A later version of the Haystack system was introduced in [9]. In this approach RDF (Resource Definition Language) is used for modeling and storing personal information. An ontology is also created for organizing,

manipulating and retrieving personal information. Agents are deployed in the Haystack system to extract and find information the user is interested in. An agent can be responsible for searching through e-mails, while another is responsible for the calendar. Since most of the code is devoted to creation and manipulation of RDF-encoded metadata, a programming language called Adenine have been built. It includes native support for RDF data types and makes it easy to interact with RDF containers and services. Adenine compiles the code straight to RDF.

2.4 CAIM

The CAIM (Context-Aware Image Management) project [1] focuses on context-awareness for supporting semantic-based image management in distributed, multimodal and mobile environments. The project is a collaboration between UiT, UiB, NTNU, and Telenor R&I. The motivation behind this project is to develop new technologies concerning context-awareness in images, and providing tools for context-aware image management.

2.5 Neste avgang

Neste Avgang (Next Departure) is a bus information service developed by a company in Tromsø called mPower [2]. Their motivation is to create services that make it easier to browse route information. Neste Avgang provides route information the user is interested in, by letting the user choose his own favorite bus stops from a map. By doing this the user can select the bus stops he uses the most. This makes irrelevant route information disappear. The service is web based and can be reached from an ordinary web browser, the system is also developed for mobile web browsers. The system focuses on listing the next departures for a given bus stop.

2.6 TravelMagic

TravelMagic is a web-based planning tool developed by Data Grafikk [3]. The system requires two input parameters from the user; where the user is traveling from, and where the user is going. The user must type this information manually into the system. It then finds bus stops the user can travel from and to. The system also support transitions, i.e. if the user must change bus during the travel to get to the specified destination. It also calculates when the user should start walking to the bus stop he is traveling from, and when he will arrive at the specified destination.

[1] CAIM project webpage: http://caim.uib.no
[2] Neste Avgang webpage: http://www.reiskollektivt.no/tfk/main.php
[3] TravelMagic webpage: http://rp.tromskortet.no/scripts/travelmagic/travelmagicwe.dll/p

2.7 Trafikanten

Trafikanten is a mobile application developed for Android and iPhone. Its goal is to provide users with real-time traffic information based on the user's location [4]. The application provides information about the next departures for nearby metro and train stations. Trafikanten uses the current location as a context-factor when extracting route information.

2.8 Discussion

In this chapter we have mentioned some projects and systems that are related to PIA, and some that are related to the route information system we are designing and implementing.

SIS, SEMEX, CAIM, and Haystack are designed to handle information fragmentation, and to make information previously accessed easy to re-find. These systems have a common limitation, they are developed to increase the PIM on a single computer. With PIA, on the other hand, the focus is to build a middleware platform (personal cloud) that makes personal information residing on various devices and services available through a single channel. This makes it easier to create new services by combining different information resources. It also means that we are able to reach information residing on different devices and services through a single platform.

Trafikanten, TravelMagic, and Neste Avgang are existing systems that provides route information. These systems share the same limitation, they do not utilize the most important context-factors when acquiring route information. Trafikanten uses some of the user's context (current location) when providing the next departure times for the user. TravelMagic requires the user to map his current context manually into the system by typing the locations he is traveling from and to. Neste Avgang is a system that must be configured by the user before being used. The user must first select his own personal set of bus stops before he can see the next departures.

PIM is the main motivation behind all of the systems we have mentioned in this chapter. The concept of PIM is concerned with how we organize our personal information. The organization determines what level of PIM we have. Our goal when designing our system is to increase the user's PIM. If our system is easier to use and requires less time than using ordinary PDF– and printed timetables, we have succeeded. Focus will also be to use context-factors when extracting route information.

[4] Trafikanten webpage: http://labs.trafikanten.no/applikasjoner/

Chapter 3

Requirements

In this chapter we will take a closer look on the requirements of our system. The requirements specify what functionality and characteristics our system must have if we are to reach our goal. We divide our requirements into two categories. *Functional–* and *non-functional* requirements. Functional requirements describe what the system is supposed to accomplish, i.e. what services it can provide for its user or application. A good routine for capturing functional requirements is by using *use-case diagrams*. A use-case diagram describes the interaction between one or more actors and the system itself. Since we are building a mobile application and a back-end system, we find it naturally to create two use-case diagrams. The first diagram describes what functionality the application provides for the user, this front-end use-case diagram can be seen in Figure 3.1. The second diagram describes what functionality RIS offers, we call it the back-end use-case diagram, this diagram can be seen in Figure 3.2. The use-case diagrams will be useful throughout this project. The back-end use-cases will be used when designing the RIS web-service, and the front-end use-case diagram will be used when designing the user-interface for the mobile application. The use-cases will also be used to evaluate if our system fulfill the specified requirements.

Non-functional requirements specifies the characteristics of the system, for instance performance, security, and reliability. We list these requirements in Section 3.3.

RIS relies on extracting data from different resources for fulfilling its functionality. These resource requirements are listed in Section 3.4.

3.1 System overview

The system we are building will consist of two major components, a front-end client (mobile application) and a back-end server (RIS). The client is responsible for providing route information for the user. The route information is offered by the server as a set of resources. The client sends requests containing parameters describing the user's context and what resource it needs from the server. The server then builds the resource the client requests, and returns it.

3.2 Functional requirements

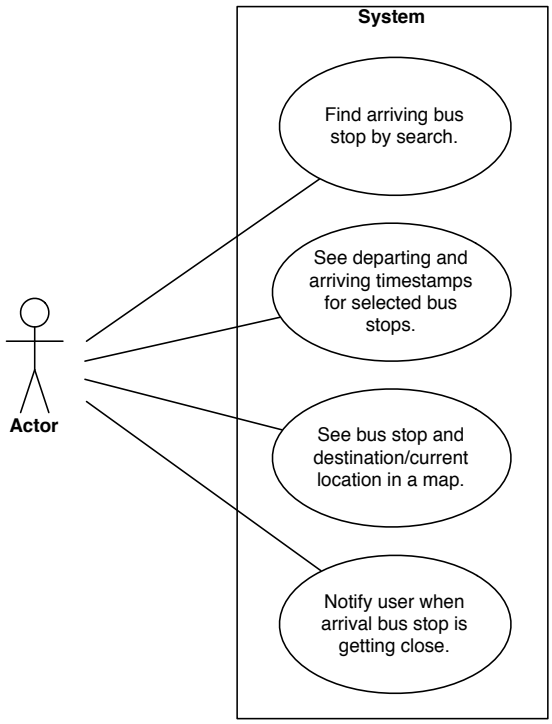

Figure 3.1: The front-end use-case diagram.

3.2.1 Client: Mobile Bus Application

In Figure 3.1 we specify what information and functionality the application shall provide for the user. In the following list we describe each use-case:

- **Use-case 1**: The first use-case shall offer a search function. The search defines where the user is going. The function should be flexible and not only support bus stops, but also addresses, and famous places and buildings. The destination defined in the search must be validated by the system, i.e. checking if it exists or not. If it exists, bus stops close to it will be returned, if not an error message should be presented. The user can afterwards pick from a set of bus stops that are returned from the server, if any. When the user has picked an arrival bus stop, the application should present nearby bus stops the user can use to get to the arriving bus stop.

- **Use-case 2**: The second use-case shall offer route information. When the user has specified the bus stops he is traveling from and to, he should be able to see the next three departure timestamps. The user shall also be able to see when the bus arrives at the destination bus stop.

- **Use-case 3**: The third use-case shall offer a map service. The map service can be used to see where the bus stop is located compared to a destination, or it can be used to see where nearby bus stops are located compared to the user's current location. The map should also illustrate where the user can walk to get to the bus stop or the destination.

- **Use-case 4**: The fourth use-case shall offer the ability to notify the user when he is getting close to the destination bus stop.

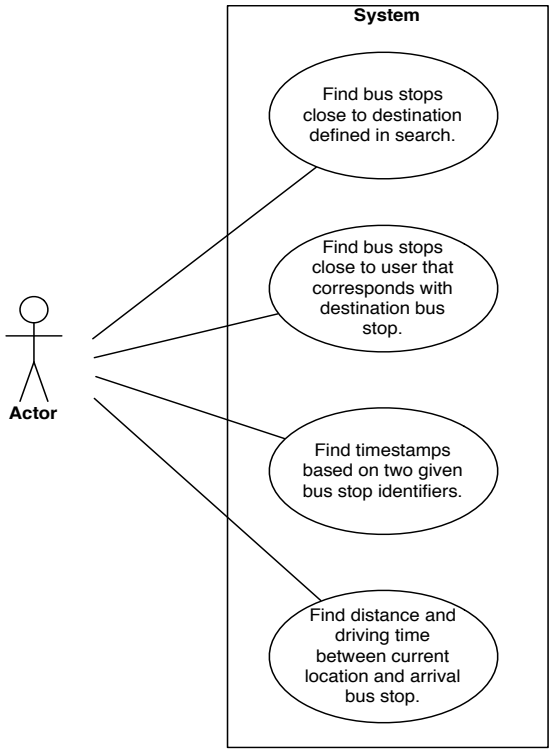

Figure 3.2: The back-end use-case diagram.

3.2.2 Server: RIS

RIS shall provide an interface offering route– and location information for the clients. The functions offered are listed in the back-end use-case diagram in Figure 3.2. The client and the server are tightly coupled since the application determines what functionality the server must provide. The RIS requirements are therefore gathered based on those for the mobile application. In the list below we describe each back-end use-case one by one:

- **Use-case 1**: The first use-case shall offer to find bus stops close to a destination defined in a search string. The destination can for instance be an address, a bus stop, or a famous place or building.

- **Use-case 2**: The second use-case shall offer to find bus stops close to the user based on the destination bus stop and the user's location.
- **Use-case 3**: The third use-case shall offer to find route information based on two specified bus stops. Routes that contain both bus stops shall be returned with corresponding departure and arrival timestamps.
- **Use-case 4**: The fourth use-case shall offer to find the driving distance and driving time based on two locations.

3.3 Non-functional requirements

In this section we list our non-functional requirements. These requirements are as important as those specified in the use-case diagrams, even though they do not concern the system's functionality. A system that is able to offer the functionality stated in the functional requirements, but lacks performance and usability, can result in not being used. We must therefore consider these non-functional requirements thoroughly when we design and implement our system.

3.3.1 Client: Mobile Bus Application

R1: Usability The application shall present simple and easy-to-understand interfaces to its users.

3.3.2 Server: RIS

R2: Accessibility The functionality offered by the RIS shall be accessible from anywhere.

R3: Data transfer The amount of data returned from the server shall be kept as small as possible.

R4: Performance Performance bottlenecks shall be addressed in our system.

R5: Platform independent The server shall not be platform dependent.

R6: Interface The server interface shall be kept as simple as possible.

3.4 Resource requirements

The server depends on being able to extract information from various resources if it is going to deliver the functionality specified in Section 3.2.2. We will now list a set of resources and describe what they should provide.

- **Translating addresses into coordinates**: Since the application shall support address search, we need a service that can translate them into coordinates. Coordinates will play an important role in our

system since they will be used when finding bus stops close to a location. The coordinates will also be used when finding the distance between two locations.

- **Route information**: The RIS must also be able to extract route information from a data source. Timestamps indicating when the bus departs from a bus stop is an essential functionality in our system. This source must also offer a list of bus stops with coordinates. The bus stops and the route information must be tied together so we can perform timestamp extractions based on two bus stops.
- **Travel information**: The RIS shall be able to offer useful information the application can present for its user, such as road distance between two coordinates, walking time, and driving time.

Chapter 4

Design

In this chapter we will present the design of the RIS and the mobile application. Design is the planning phase that lays the basis for our implementation. We are going to discuss each functional– and non-functional requirement listed in the previous chapter, and how we will fulfill them.

4.1 Architecture

4.1.1 Decomposition

Route information is stored in a database which is accessed by the RIS when requests are received from the clients. A request contains the user's context, which describes his current location and travel interests. The application and RIS are tightly coupled. We will use the use-case diagrams in Figure 3.1 and Figure 3.2 to illustrate how the system is decomposed.

When the user starts the application on the mobile device, it must find out where the user is traveling. The application starts off by asking the user this question. The user tells the application where he is going based on a search (Front-end use-case 1). The application should not rely on the user knowing the name of the bus stop he is traveling to. Instead, the system should offer different search types. The first search type is address, it offers the user to search for the address he is going to. The second search type is place, a functionality where the user can search for famous places or buildings. The last search type is bus stop names. When the user has defined the search, it is sent of to the RIS. The RIS must now find bus stops that are close to the address or the place that was defined in the search (Back-end use-case 1). The RIS finds the coordinates of the destination by using a technique called *GeoCoding*. These coordinates and the coordinates representing the user's current location, are used to find bus stops the person can travel to.

The result of the search is a set of bus stops close to the address or the place. These bus stops are presented for the user. He must now select one of the bus stops. The selected bus stop and the user's current location are the input parameters for finding out which nearby bus stops the user can depart from to get

to the selected bus stop close to the destination (Back-end use-case 2). These nearby bus stops are also presented for the user.

The user have now selected the arriving bus stop on his travel. The remaining part we need to know before showing routes and timestamps (Front-end use-case 2) is the bus stop he is departing from. The user must select one of the bus stops that are shown in the mobile application. If the user is unaware of where the bus stops are located, he can use a map which is integrated in the application to see where it is located compared to his current location (Front-end use-case 3). The application sorts the bus stops based on the distance between the bus stops and the user's location. When the user finally decides what bus stop he is departing from, we can find timestamps and routes describing the next departures. This is easy since we know the bus stops the user is traveling from and to. The two bus stop identifiers are used by the RIS to find timestamps and routes (Back-end use-case 3).

Since the fictional user of this system is unknown in the area of where he is traveling, we should provide a notification system that indicates when the user is getting close to the arriving bus stop (Front-end use-case 4). The user can activate the notification system himself. When it is activated the user's current coordinates are sent to the RIS and compared with the destination bus stop's coordinates. The time and distance left are calculated, and sent back to the application (Back-end use-case 4). When the distance comes below a certain limit, the application will tell the phone to vibrate. The user should notice the vibration and get ready to leave the bus.

4.1.2 Non-functional requirements

In this section we will discuss how we handle the non-functional requirements for our mobile application and the RIS.

The Mobile Application

The bus application must present its functionality in a sleek and simple manner (R1). The user-interface must be logical and easy to understand. It is important that the application always notifies the user on what is going on in the background. When the application is loading data from the RIS, the user should know about it. To notify the user about background activities we should show and hide activity indicators, for instance a spinning wheel. It is also important to follow common design patterns used in other applications that are developed for the selected platform. For example, on the iPhone platform, it is good practice to notify the user when the application is transferring data over the network. This is done by simply setting a boolean value to true when downloading, and false when not downloading. This will show and hide a dedicated activity indicator created for this purpose. By doing this the user naturally understands how the user-interface works. Our goal is to make the application faster and easier to use compared to existing systems.

RIS

A mobile device running the bus application is dependent upon receiving route data from the RIS. Making this possible wherever the client is located is done by giving the RIS a unique IP address (R2), i.e. placing it on the Internet. This solution requires that the mobile device is able to connect to the Internet. It is not considered a problem since most devices are equipped with network technologies such as EDGE and UMTS. Another solution could have been to store route information locally on the client. If we chose this solution we would have to store all the route information locally on each device. The storage capacity would not have been an issue since most devices today often have gigabytes of local storage available. Consistency on the other hand would have been an issue, since changes occur in the timetables, and bus stops are added and removed. We avoid this issue by always accessing a single updated data source each time we request route information. Another benefit with this approach is that most of the overall logic resides on the RIS. This makes the task of developing clients easy and does not require that much effort. The client we develop is more or less a thin-client[1].

Transferring data between the RIS and the mobile application requires network activity (R3). Network activity is often charged per MB, and can in some cases be expensive. Our goal should therefore be to keep the data traffic as small as possible. On mobile devices bandwidth varies depending on network coverage and signal strength. This means that the time it takes to transfer data depends on where we are located. Data size also plays a role in how long it takes to transfer it. With variable bandwidth it is good practice to keep the data amounts as small as possible. We can achieve this by sending only what is necessary, and choosing a data-interchange format with small overhead. XML (eXtensible Markup Language) is today a well-known standard, and often preferred for transferring data between systems and for organizing and storing data [4]. In XML, values are stored in tags, these tags causes more bytes to be transferred between the client and the server, i.e. more overhead. A format that is more lightweighted, when it comes to size, is JSON (JavaScript Object Notation)[2]. JSON stores values in a structured manner. A data collection is stored as name/value pairs, with no tags between the values. This means less overhead compared to XML. Since this format is cheaper it shall be used in our implementation.

When we have implemented the RIS, experiments will be conducted to benchmark the performance. We will measure how long it takes to build each resource that it offers, and eventually conduct lower-level measurements to find the cause of any bottlenecks (R4). We will suggest optional design solutions if necessary, but not implement them. Our main focus in this thesis is the system's functionality. Work on optimizing the system will be left for future work.

The main logic in our system resides on the server. If we want to utilize this advantage and create mobile applications for several platforms, it must be platform independent (R5). Our system should therefore offer

[1] A computer that depends heavily on another computer. There is no data-persistence on a thin-client, most of the logic resides on another computer.

[2] JSON (JavaScript Object Notation) homepage: http://json.org/

its functionality through a web-service. RIS will follow this approach and offer its functionality through a RESTful (REpresentational State Transfer) API. REST is a software architecture for distributed hypermedia systems such as the World Wide Web [8]. We will provide our own resources by extracting and merging data from different locations and giving them our own global identifiers. The clients will use the HTTP protocol when communicating with the web-service. A request contains the identifier of a resource and the required parameters needed to create it. The web-service will return the resources as JSON documents.

The web-service offered by the RIS should be simple and logical (R6). A description will be provided through a documentation listing the different resources available, and the parameters and datatypes needed to create them. We will focus on decoupling the system's functionality when designing the interface.

4.1.3 Overview

The architecture for our system and its components can be seen in Figure 4.1.

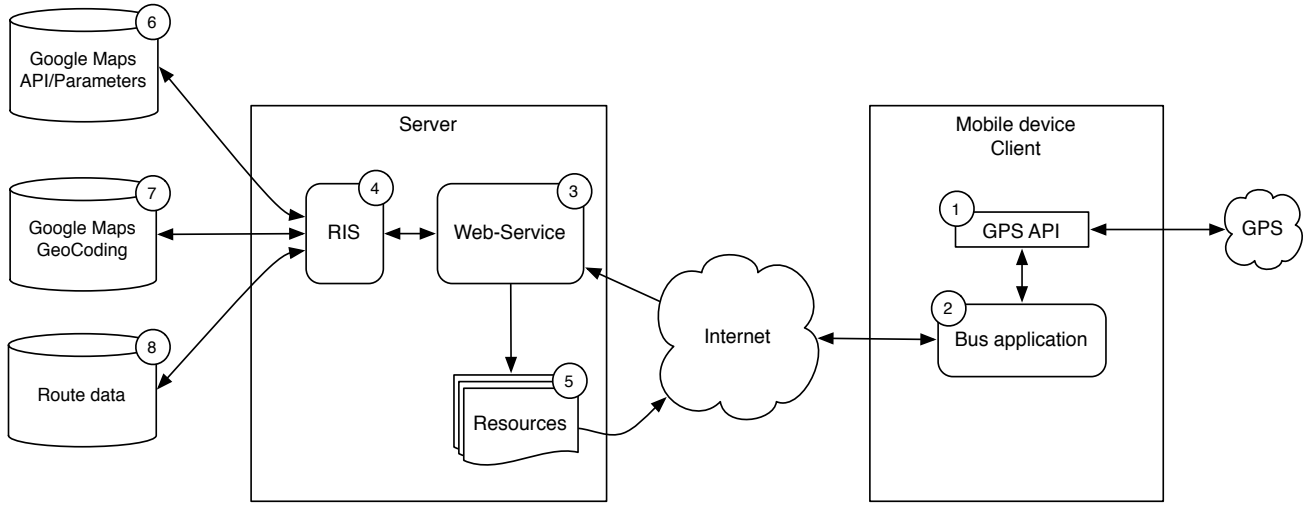

Figure 4.1: Components overview.

GPS API (1): Coordinates describing where the user is located at any given time is provided by an API that communicates with a GPS hardware component. The support for GPS must be available on the mobile platform since the RIS requires the coordinates as a parameter for creating some of the resources.

Bus Application (2): The bus application acts as the client in our architecture. It extracts route information from the RIS by communicating with the web-service.

Web-Service (3): The web-service is the interface of the RIS which the clients communicate with. It receives requests and passes them to the RIS-core. The web-service returns a resource as a response to the request.

RIS (4): The RIS (Route Information Server) is responsible for extracting data from the resources and creating new ones. The RIS offers its functionality through the web-service.

Resources (5): A resource is an output file the mobile bus application can read generated by the RIS. It may contain route information or information about nearby bus stops. It depends on what resource the client requests. The resource is a structured JSON document.

Google Maps API/Parameters (6): This component is accessed by the RIS to find the distance between two coordinates.

Google Maps GeoCoding (7): This component is accessed by the RIS to translate addresses into coordinates and vice versa.

Route Data (8): This component is a database containing information about bus stops, routes, and timestamps. Queries are issued by the RIS when it needs timestamps or information concerning bus stops.

4.2 System components

In this section we will give a more detailed discussion of our system. It is important to have a well designed architecture before we start implementing our system. A good design means locating and designing a solution to all the risks that may impair the process of implementing the system. With a good design in place, the chance of discovering new risks are reduced. A bad design could lead to more time being spent on new iterations caused by problems being found in later phases.

4.2.1 GPS

The GPS (Global Positioning System) component makes it possible to pinpoint a user's current location. Most mobile devices today use a system called A-GPS (Assisted-GPS) [10]. In A-GPS, dedicated computational servers are used for calculating and finding out where the user is located. Since signals often are weakened by buildings and structures, it requires more calculations to find the accurate position. Most mobile devices do not have enough processing power to solve these calculations within a reasonable time alone, dedicated servers are therefore used to perform the calculations. In our architecture we use GPS on the client to pinpoint the user's location. The coordinates are required as parameter in some of the functions offered by the RIS.

4.2.2 Mobile application

The mobile application utilizes the functionality provided by the RIS. It is responsible for displaying information of interest to its users. It must also handle the user's context. The user's context describes what situation the user is in at a given moment. In this thesis we consider the context of a person that wants to travel by bus, but does not know where the bus stops are located, what route to use, and so on. We must therefore map the context into the application. The user's location is found by the GPS. Specifying the bus

Figure 4.2: User-interface illustrations of the mobile application.

stops the user wants to travel from and to, requires manual input. The application will help the user in this decision-making by showing illustrations in maps, showing the distance between the bus stops and the current location, and by calculating the time it takes to walk from a location to another. These inputs are simple since the user only has to select one of the suggested bus stops. The only context-factor the user have to manually type into the application is the destination. This context-factor can not be extracted in

an automatic fashion without input from the user.

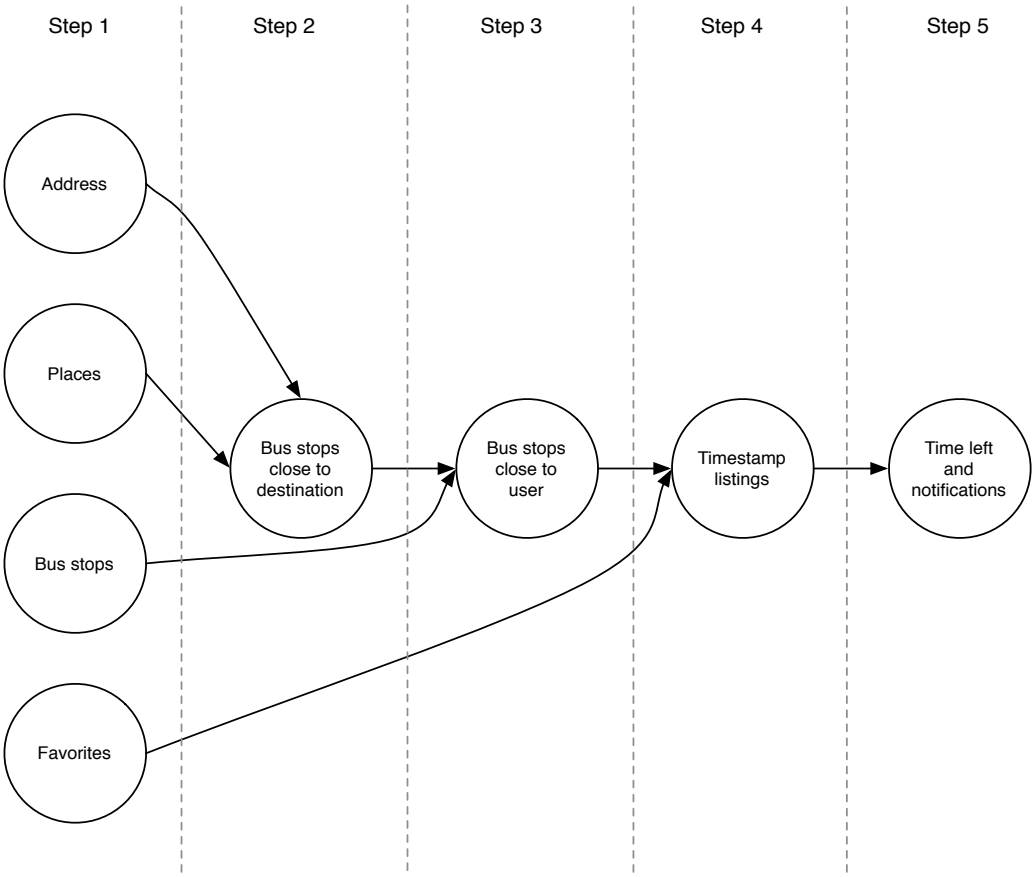

Figure 4.3: All possible usage patterns.

User-Interface

The user-interface for the mobile application is an important part of the system. This is what the system looks like for the user. A simple, logical, clean, and efficient design is the goal when designing it. In this thesis we started off gathering the requirements for the mobile application, they were captured in the use-case diagram in Figure 3.1. We can now map these requirements into sketches which illustrate what information and functionality the application presents and offers the user. When we know the requirements for the application and how we want it to look, we can start implementing other components, such as the RIS which offers information the application needs. The user-interface illustrations are listed in Figure 4.2. In Figure 4.3 we show all the different usage patterns. A description to Figure 4.2 is given in the following list.

1. When the application is launched on the mobile device, the user is asked where he is going. The user can choose between three different search types; address, place, or bus stop. The user can also choose from a favorite list containing previously used distances. The search string defined by the user

depends on what the user knows about the destination. If the user knows the name of the bus stop, the application will change window to number 4. But if the user only knows the address or the name of the building he is going to, an arrival bus stop must be chosen. This is done in window number 3. If the user choose to use a favorite distance, he will go directly to window number 5 which list the next timestamps.

2. The second window illustrates real-time suggestions by the application. When the user starts filling in the search string, the application will automatically list suggestions. This functionality works for all three search types.

3. The third window lets the user select the arriving bus stop. The application lists a set of bus stops that are close to the destination defined in the search. The list is ordered by the distance between the bus stops and the destination. The application will also tell the user the time it takes to walk from the bus stops to the destination. A map is also provided, and the user can simply click the blue disclosure button to view it. Pins are placed on the map to illustrate where the bus stop and the destination are located.

4. The fourth window appears when the user has selected the arriving bus stop. This window is identical to window number 3, but instead of showing bus stops close to the destination, we list bus stops close to the user. The items listed are bus stops that contain routes that will take the user to the selected destination bus stop. The user can also here see the distance between the current location and the bus stops, and how long it takes to walk. A map window is also available, which shows where the bus stop is located and the user's current location.

5. The fifth window appears when the user have selected the nearby bus stop he wants to travel from. This window shows the user which routes he can use to get to and from the selected bus stops. The three next departure times for each route are also included, it is listed when the bus arrives to the destination bus stop as well. The user can choose to save the travel combination in a favorite list for later use, by pressing the add-to-favorite button. A notification service provides an indicator telling the user when he is close to the destination bus stop. The service is activated by pressing the activate-notification-service button.

6. The sixth window is used for notifying the user that he is close to the destination bus stop. The application will regularly compare the user's current location with the bus stop's location. When the user approaches the arriving bus stop the application will start alerting the user by making the phone vibrate. The user can also see the remaining travel-distance in this window.

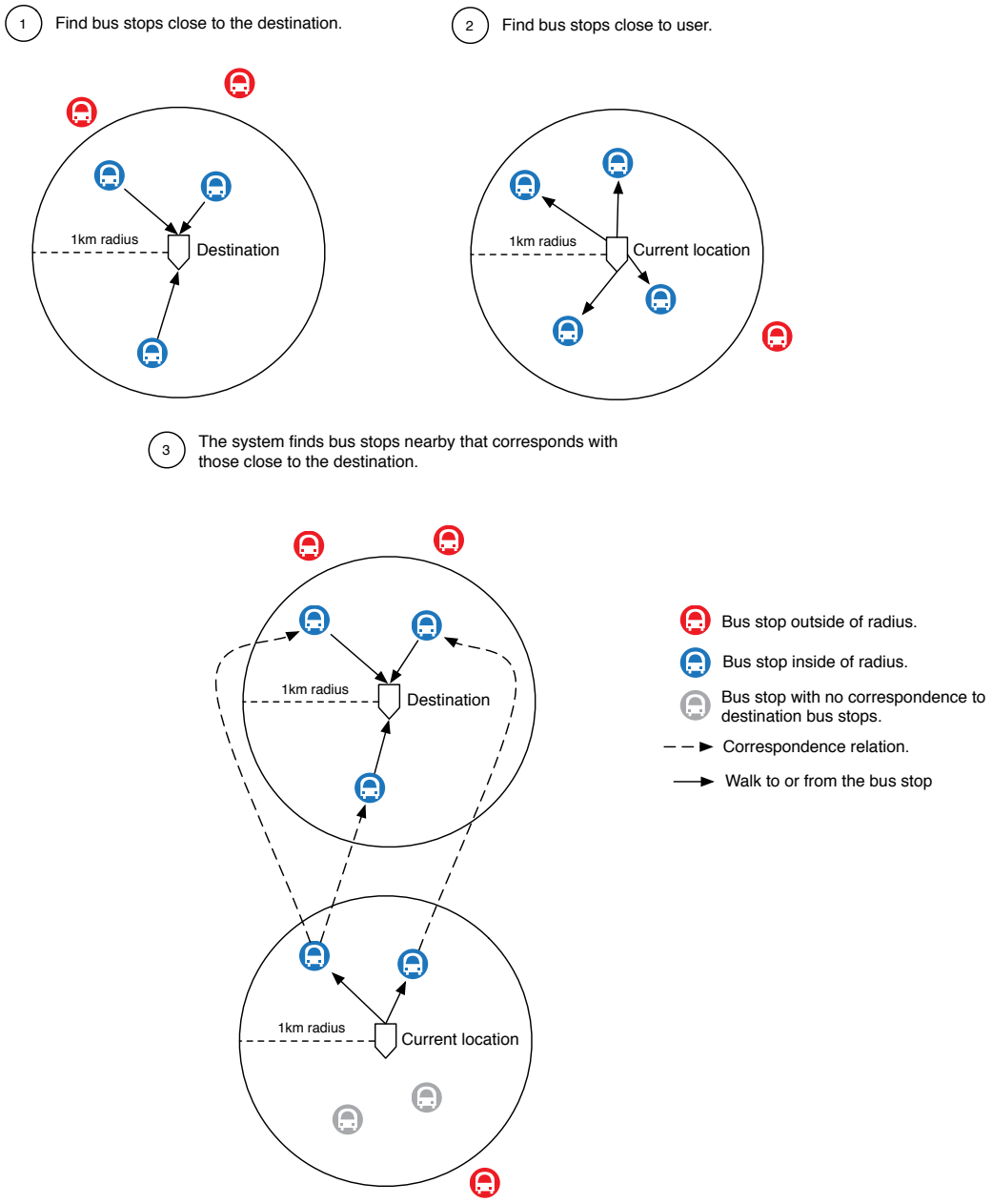

Figure 4.4: How the system finds bus stops the user can travel from and to.

4.2.3 RIS

The RIS is the most important component in our architecture. It is the crossroad between almost all the other components. It offers an interface to its clients. Through this interface the clients are able to find documents containing data they need. The RIS extracts data from a set of resources. New resources are composed from these extractions and returned as structured JSON documents to the mobile application.

Web-Service

The web-service provided by the RIS, offers access to a set of resources. The client must specify in the request what resource he is interested in. Each resource requires different parameters. They are used by the RIS to extract information and for creating the resource.

The requirements for the RIS are specified in the use-case diagram in Figure 3.2. The resources provided by the web-service is listed beneath.

- **Bus stops close to destination**: The first resource offered by the web-service is called *bus stops close to destination*. The logic behind this resource is illustrated in Figure 4.4. The resource requires two parameters, the user's current location and a search string defining a destination. A destination is in this case an address or a place the user wants to travel to. If a destination bus stop is selected instead, there is no need for this resource since we know which bus stop we are going to and its coordinates. (1) When this function is initiated, the RIS must first of all find bus stops that are close to the destination. This is done by drawing a circle around the destination with a given radius, and determining which bus stops that are within. (2) The second step is to find bus stops that are close to the user, this is done in the exact same way as in the previous step. (3) The last step is to find out if any nearby bus stops have routes that will take the user to any of the bus stops located close to the destination. Bus stops close to the destination that have correspondence with nearby bus stops, are added to the JSON resource.

- **Bus stops close to the user**: When the user has decided which bus stop he wants to travel to, the next resource can be used, *bus stops close to the user*. This resource is generated based on two parameters, the user's current location, and the selected arrival bus stop. The logic of finding bus stops close to the user's location is the same as illustrated in Figure 4.4 step 2. The RIS will find bus stops close to the user's current location that correspond with the destination bus stop specified in the parameters. These bus stops are added to the JSON resource.

  This resource could have contained route and timestamp information. If we have piggybacked it, we could have reduced the total number of requests sent to the server by one. The drawback of this solution would have been the risk of data not being accessed by the user. E.g. if the system presents 10 bus stops for the user, the probability of him being interested in timestamps for all of them is very small. In most cases the user will only be interested in the bus stop which is closest to his current location. The result of this overhead will also increase the amount of bytes being transferred over the wire, which means more money being spent on data traffic. In our implementation we will use an extra request for accessing route information. By doing this, we will only ship information the user is interested in.

- **Departures corresponding to selected bus stops**: This resource is used for finding departure and

arrival timestamps based on two bus stops. The resource is called for when the favorite function is used, and when the user selects one of the bus stops that are nearby.

- **Distance and time between locations**: This resource is utilized when the user activates the notification service in the application. The application sends two coordinates, current location and destination, to the server. It then finds the distance and the time it takes to drive between the locations. The values are returned back to client.

4.2.4 Resources

A resource is the result of a request. It contains structured data that can be parsed by the client. In our system we present our resources as JSON documents. Each call to the RIS results in a resource being built and returned to the client. The resources offered by the RIS are listed in Section 4.2.3.

4.2.5 Google Maps API/Parameters

This component is a web-service offered by Google. It is used for finding the road-distance between two coordinates. It is also used for finding the time it takes to drive or walk between two locations. We also use this web-service when we create the graph containing coordinates that illustrate where the user should walk. Instead of using this web-service we can use air distance for all our calculations, but this will lead to inaccurate measurements since the bus drives along a road and not in a straight line between the bus stops.

4.2.6 Google Maps GeoCoding

Offering address-search in our mobile application means we have to translate addresses into coordinates. We need the coordinates before we can start finding bus stops close to an address. The process of converting an address into coordinates is called GeoCoding. In our implementation we will use Google's GeoCoding web-service for handling this task.

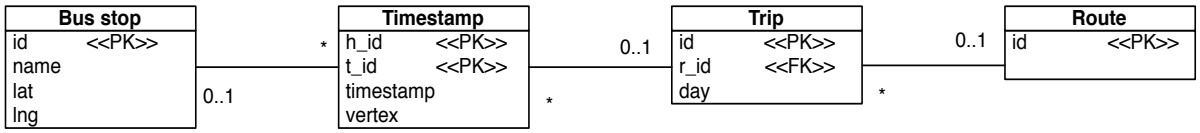

Figure 4.5: Database model showing the structure of the database holding the route information.

4.2.7 Route data

Information concerning bus stops and timestamps are stored in a database. The database model is listed in Figure 4.5. The database is used for several things in our system, but most importantly extracting bus stops

and finding timestamps. A problem we base our data model on is being able to extract timestamps based on two known bus stops. A solution to this problem is to create graphs that connect bus stops together. Each vertex in the graph contains these values.

```
<Busstop ID, Graph ID, Timestamp, Vertex number>
```

This is a simple example of a graph.

```
[<1, 1, 08:02, 0>, <2, 1, 08:05, 1>, <3, 1, 08:14, 2>]
```

This graph contains three vertices. A vertex in the graph tells when the bus arrives at a bus stop. The vertex numbers shows us what position the bus stop has in the graph. The graphs are given a unique ID, and are stored in the trip table with a day flag indicating when the graph is valid during the week. Each graph is also connected to a route. Based on these graphs we can find all the trips where the two bus stops occur. If we find graphs where both bus stops occur, we know that the bus travels between the bus stops, but we do not know if it is the right direction we are interested in. Finding the direction we are interested in can be done by comparing the vertex numbers. For instance, if we want to travel from Bus stop 1 to Bus stop 3, there must be one or more graphs containing both bus stops, and the vertex number of Bus stop 3 must be higher than the vertex number of Bus stop 1. If this is the case, the graph can be used for extracting valid timestamps based on those bus stops.

If we have not used graphs as a glue between bus stops, we could not have found timestamps based on where we are going and where we are going from. The database stores a large amount of graphs that are easily accessible through simple queries.

4.3 Summary

In this chapter we have found an architecture that we believe will fulfill our requirements. We have analyzed the requirements of the mobile application and suggested how the user-interface should work. Based on these requirements, we have also found what resources the RIS must offer through its interface. The RIS will offer its functionality through a RESTful API. We have also discussed how the resources should be provided by RIS, and came to the conclusion that JSON would be the best solution for our system. We have also found which services we will extract data from when building our own resources. We will use Google Maps web-services diligent to GeoCode addresses into coordinates, and to calculate distances and time. We have also found a way to store route information that makes it possible to find timestamps and routes based on two bus stop identifiers. We found that a relational database should be designed for storing and accessing route information.

Chapter 5

Implementation

5.1 Implementation environment

5.1.1 RIS

The RIS is implemented using PHP (PHP: Hypertext Preprocessor). The PHP scripts are deployed on a web-server hosted by PRO ISP[1]. It's functionality is made available through a web-service which accepts GET requests from the mobile application. The main reason why we choose PHP as the server-side language is because of personal experience, but also because of its functionality. With PHP it is also easy to offer a RESTful API.

5.1.2 Mobile Application Prototype

The mobile prototype application is implemented on the Apple iPhone platform. The source code is written in Objective-C. We have compiled and tested our code on iPhone OS 3.0. For debugging we have used both the iPhone simulator on a Mac and a real iPhone. The main reason why we chose to write the client for the iPhone is because of the good SDK (Software Development Kit) and the number of developer communities online. Personal interests also played a role in this decision making.

5.1.3 Route information

The route information is stored in a MySQL database, also provided by PRO ISP. We choose to store route information in a database because it is much easier to send queries to a database than to parse structured files such as XML.

[1]PRO ISP homepage: http://www.proisp.no

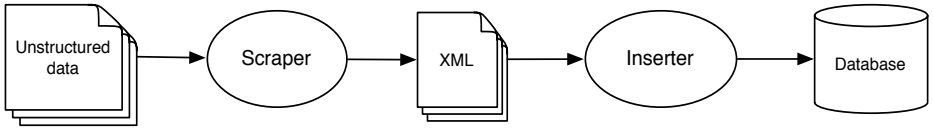

Figure 5.1: The process of generating and inserting route information into the database.

5.2 Implementation details

5.2.1 Route information

In our system we store route information in a database. The database contain real-life bus stops and route information from Tromsø (route 20, 21, and 24). We have inserted the data into the database ourselves. We got the initial data source from scraping PDF files and online route systems. The process of generating the route information is shown in Figure 5.1. A parser was written in Python creating XML files that later would be inserted in the database by a "inserter-script" written in PHP. By using real-life bus stops and route information we are able to conduct real-life experiments.

The code in Listing 1 shows the main Python script that creates the XML files for the inserter-script.

```
1 departures = parserFuncs.findDepartures("avganger_0_20_dx67.txt", 20)
2 busstops = parserFuncs.findBusstops("navnholdeplass_0_20_dx67.txt", departures)
3 trips = parserFuncs.findTrips(departures, busstops)
4 parserFuncs.toXML(trips, "20", "0", "20_0_dx67.xml", "dx67")
```

Listing 1: Main parser script implemented to create XML files for the inserter-script.

The script starts by finding the timestamps when the bus leaves the first bus stop in the route. The next step is finding the name of all the bus stops in the route. When the bus stops are found we can create the trips. A trip is a ride from the starting bus stop to the destination bus stop. A single trip contains one timestamp for each bus stop indicating when the bus arrives. The trips are finally stored into an XML file ready to be parsed by the inserter-script. Listing 2 shows the output generated by the Python parser.

The last step in the process, is inserting the route information into the database. This is done by the inserter-script. The inserter-script simply reads the XML-files generated by the scraper-script, and inserts the trips into the database.

```xml
<root day="dx67" retning="0" rute="20">
        <trip>
                <timestamp id="0" navn="Malmvegen snuplass" time="06:00"/>
                <timestamp id="1" navn="Malmvegen sor" time="06:01"/>
                <timestamp id="2" navn="Granittvegen" time="06:01"/>
                <timestamp id="3" navn="Kvartsvegen" time="06:02"/>
                <timestamp id="4" navn="Gneisvegen" time="06:03"/>
                <timestamp id="5" navn="Krokensenteret" time="06:04"/>
                <timestamp id="6" navn="Krokelva" time="06:04"/>
                <timestamp id="7" navn="Myrland" time="06:05"/>
                <timestamp id="8" navn="Lunheim vest" time="06:06"/>
                <timestamp id="9" navn="Austheim nord" time="06:06"/>
                <timestamp id="10" navn="Austheim" time="06:07"/>
        </trip>
</root>
```

Listing 2: XML data generated by the Python parser script.

5.2.2 RIS interface

The RIS makes a set of resources available to the clients through a web-service. The web-service we have implemented is of the RESTful kind. A RESTful web-service uses HTTP to transfer the requests, and often implements the HTTP methods; POST, GET, PUT, and DELETE. Our web-service implements the GET method only. The reason behind using GET is simple, the RIS is a read-only system, i.e. clients cannot send data to be stored, they can only request and read resources that are generated by it. The resources provided by the web-service, the API, is shown in Listing 3.

Resource name	Parameters	Description
busstops.close.to.search	searchType(String) searchString(String) currLat(Float) currLng(Float)	Find bus stops close to a place or an address based on a search string and the user's current location. A JSON document containing these bus stops are returned.
busstops.nearby	busStopId(Integer) currLat(Float) currLng(Float)	Find bus stops close to the user's current location that have routes that correspond with the destination bus stop. A JSON document containing nearby bus stops are returned.
timestamps.from.to	toBusStopId(Integer) fromBusStopId(Integer)	Find routes and timestamps based on two bus stops. A JSON document containing routes and timestamps are returned.
distance.between.coordinates	currLat(Float) currLng(Float) destLat(Float) destLng(Float)	Find the driving distance and the driving time between two coordinates. A JSON document containing these two values are returned.
path.between.coordinates	startLat(Float) startLng(Float) stopLat(Float) stopLng(Float)	Find the road path between two given coordinates. A JSON document containing coordinates are returned.

Listing 3: Table showing the web-service API.

Debugging the resources throughout the project have been done by sending requests with a web-browser to the web-service. The output of the requests, i.e. the JSON documents, have been validated to check if they contain the correct information. An example request asking the RIS to find bus stops close to an address is shown in Listing 4.

```
http://localhost/?resource=busstops.close.to.search&searchType=address
&searchString=Storgata+tromso&currLat=69.683235&currLng=18.977718
```

Listing 4: Example request sent to the web-service asking for a resource.

A small piece of the output generated by this request can be seen in Listing 5. We can see two bus stops that are found close to the address defined in the search. The RIS extracts information about the bus stop from

the database, while it uses the Google Maps API/Parameters web-service to find the distance and walking time.

```
[
    {
        "id": "2",
        "name": "Havnegata H1",
        "lat": "69.651993",
        "lng": "18.9601",
        "distance": "0.2",
        "wtime": "3 mins"
    },
    {
        "id": "19",
        "name": "Wito",
        "lat": "69.649597",
        "lng": "18.955549",
        "distance": "0.2",
        "wtime": "2 mins"
    }
]
```

Listing 5: Sample taken from a JSON busstops.close.to.search-resource.

5.2.3 Map service

The MapKit framework have been used for showing maps and illustrating points of interest in our iPhone application. The MapKit framework lack one important functionality; highlighting the road between two coordinates. This functionality is central when illustrating the shortest road-path between two locations. To solve this issue we had to include another resource (path.between.coordinates) in the RIS. The resource is built based on two coordinates, the user's current location and the destination. By using the Google Maps API/Parameters web-service and the two coordinates, we are able to extract a set of coordinates that can be used for drawing a graph on a map. These coordinates are included in the resource.

5.2.4 The notification service

The notification service is designed to help the user when traveling with the bus. The user can constantly see where he is located compared to the destination bus stop. When the notification service is activated the application will send a request to the RIS asking for the time and distance left of the travel. In our

implementation we send this request every fifth second. This solution is not optimal since it requires network traffic, and the more requests the user sends the more it will cost. The network latency is also a drawback since the application gets more sensitive when the user gets closer to the destination. The only service providing this data is Google's API/Parameters-web-service. The optimal solution would have been to find this information through the native API's provided by the iPhone platform, but since this functionality currently is missing, we have to communicate with the RIS instead.

The distance and time returned from the RIS is presented in the user-interface but also used to determine when to start sending vibration notifications to the user. When the user is less than 200 meters away from the bus stop, the application will vibrate every fifth second. The user should notice the vibrations and start thinking about the arrival.

5.2.5 Prototype user-interface

In this section we have included some screenshots taken from a real-life usage scenario. The screenshots are taken from a physical iPhone device. This is the scenario:

> *A user is located at the University campus in Tromsø, and is headed for Storgata (a shopping street in the city). The user is unfamiliar when it comes to traveling by bus. He does not know where the bus stops are located, which route he should use, and where he should get off the bus.*

To solve these issues he can use our mobile bus application. The user starts by launching the application from the home screen (Figure 5.2). He then fills in the address into the search field (Figure 5.3 and Figure 5.4). When the user has initiated the search, a result containing bus stops close to Storgata is returned (Figure 5.5), and he picks Wito as the arrival bus stop. The next step is to choose a bus stop close to the user (Figure 5.6). A map service can be used to see where the bus stops are located compared to the user's location and the destination defined in the search (Figure 5.7 and Figure 5.8). The user selects UiTø/ISV as the departing bus stop since it is the closest one. Timestamps based on the selected bus stops are now listed (Figure 5.9). He can activate the notifications service to see what is left of the travel, and to receive notifications on when he is close to the bus stop (Figure 5.10). The travel distance can also be stored for later use, these distances are available in the favorites list (Figure 5.11).

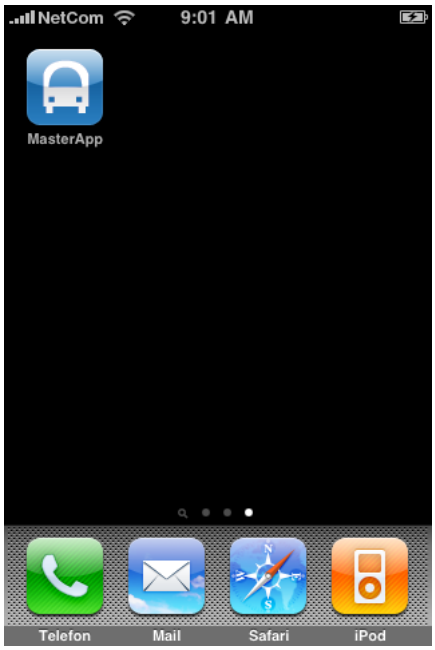

Figure 5.2: The iPhone home screen.

Figure 5.3: The welcome screen.

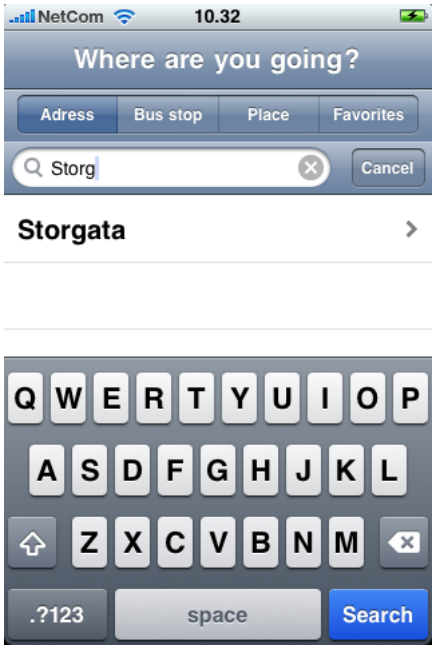

Figure 5.4: Real-time suggestions.

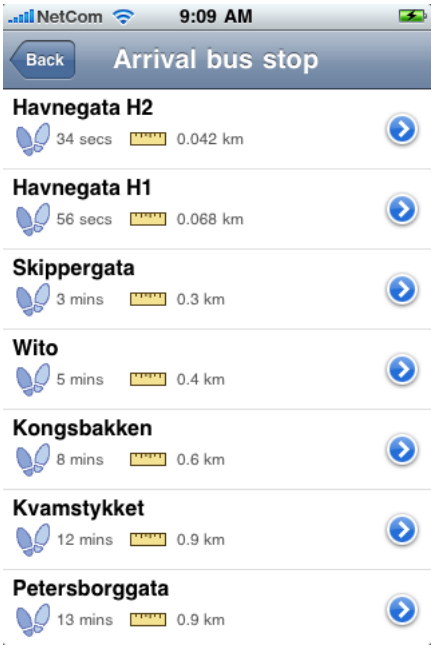

Figure 5.5: Bus stops close to destination.

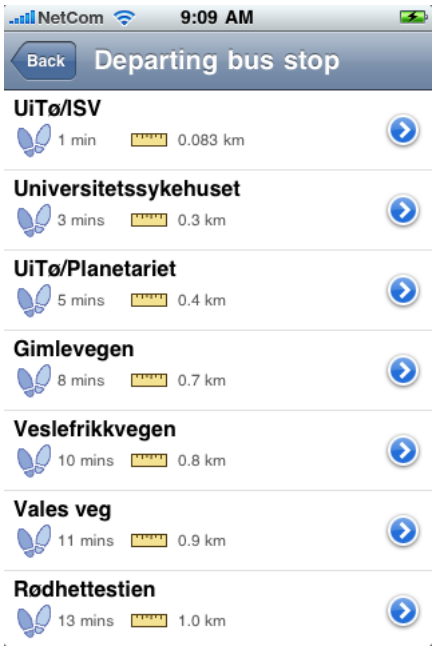

Figure 5.6: Bus stops close to user.

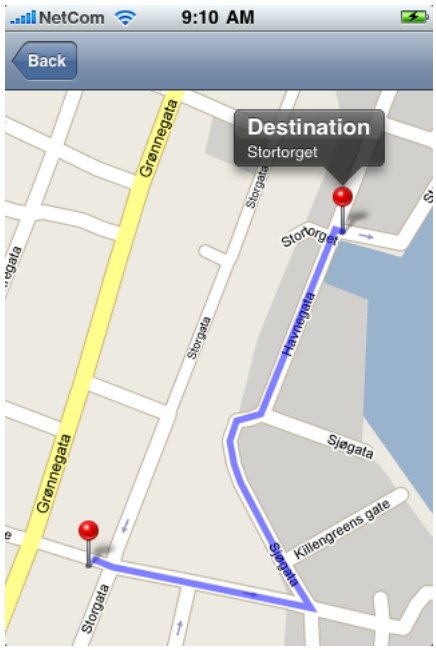

Figure 5.7: Map illustrating bus stop and destination.

Figure 5.8: Current location and departing bus stop.

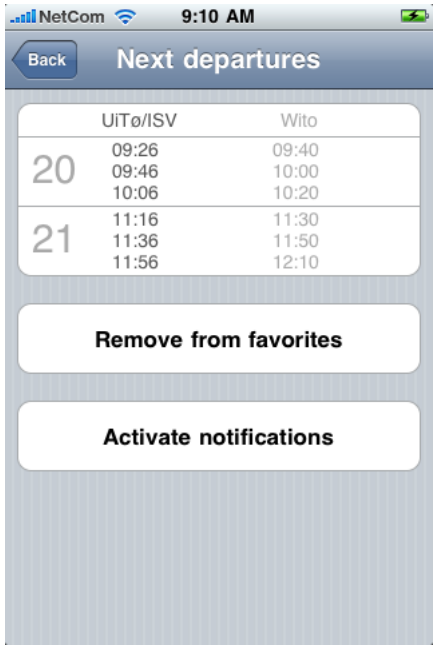

Figure 5.9: Next departures for each route.

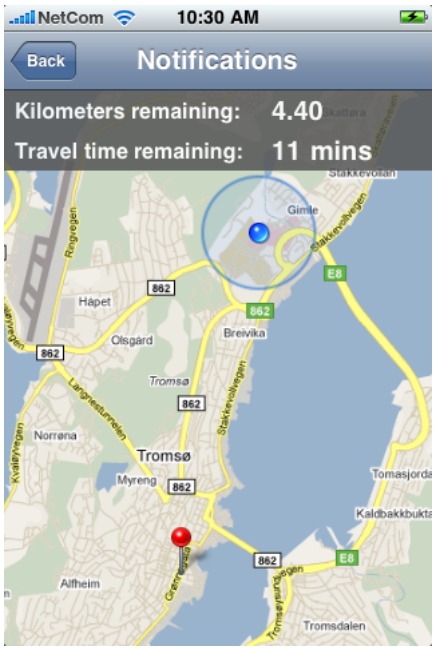

Figure 5.10: Notifications activated.

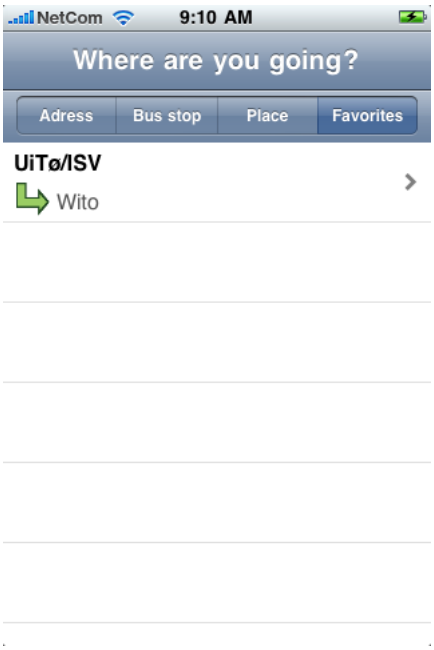

Figure 5.11: Stored favorite travel distances.

Chapter 6

Evaluation

This chapter will give an evaluation of the prototype mobile application and the RIS. We will test the implementation against the functional– and non-functional requirements to determine success or failure of our design and implementation. We will also suggest how a personal cloud, as suggested in [2], can make it easier for us to build a service providing the same functionality as the RIS.

6.1 The evaluation process

Evaluating the design and implementation is a process of comparing the end result with our goals. To reach our goals we need our system to fulfill all the requirements in the specification (in Chapter 3). A requirement is a capability or property the system must have for reaching a goal. We will therefore use the functional– and non-functional requirements to determine if we have reached our goals. First, we will identify the correct measurements for our system. Then, we must determine what results we consider successful. We can then present the results of our tests and experiments with an evaluation.

6.1.1 Measurements

Before presenting the experiments and results we will discuss how we can measure the different functional– and non-functional requirements, and how we can determine a successful result.

The functional requirements

The functional requirements for our system are described in Chapter 3.2. These requirements were gathered and put into use-case diagrams, one for the mobile application, and one for the RIS. The use-case diagrams were later used for designing the RIS and the mobile application. To determine wether the API, provided by

the RIS, delivers the functionality it promises, we will conduct an *acceptance test*[1] by using the prototype mobile application. The system will be evaluated based on real-life test-cases. The output of the tests will be compared to what we expect from the system. The mobile application will also be tested and evaluated by using the user-interface sketches illustrated in Chapter 4.2.2. By comparing the prototype implementation of the mobile application with the sketches, we are able to see if we reached our user-interface goals.

Non-functional requirements

The non-functional requirements for our system are captured and described in Chapter 3.3. These requirements are also important since they often determine how the system behaves. Some of the non-functional requirements can not be tested since they do not have a measurable unit. These requirements will instead be discussed one by one. In our system, these requirements are; usability, accessibility, platform independence, and the interface. The remaining requirements, performance and data size, will be tested because we are able to measure time and data-size. The performance experiment will be conducted on each of the methods offered by the RIS. Further low-level testing will also be done to address time consuming statements, if any. The data size experiment will be performed to investigate the difference between XML and JSON, and to back up the decision we made in the design phase.

6.1.2 The experiments

Since the experiments we are going to conduct on the system are stated, we can list them and define what we consider a successful result:

- **Acceptance test**: To determine if the system fulfill all the functional requirements, we will conduct some real-life test-cases. If the output of all the test-cases are as expected, i.e. conforming with the requirements, we have succeeded.
- **Comparing the user-interface**: To evaluate the application designed for the iPhone, we will compare the user-interface sketches with the application's user-interface. If the sketches conform with the application, we have suceeded.
- **Performance experiment**: To evaluate the performance of the system, we will measure the time it takes the RIS to build the resources. Our intension behind this experiment is to address where we have drawbacks when it comes to performance. An acceptable build time for each resource depend on how long the user accepts waiting for the application to change state. J. Nielsen mentions three important limits in [12]. The first limit is 0.1 second; this response time means the system is reacting instantaneously. The second limit is 1.0 second; this response time is noticeable by the user and may interrupt the flow. The third limit is 10.0 seconds; this is the limit for keeping the user's attention

[1] Acceptance testing involves running real-life test-cases on the completed system. The output of each test is compared to the expected result.

focused on the dialogue. If longer delays occur, the user will want to perform other tasks while waiting for the application to finish. Performing other tasks is currently not possible on the iPhone because the lack of multitasking (iPhone OS 3.0). Based on Nielsen's limits we can consider response times below 10 seconds as successful results. Nielsen also points out that process indicators such as spinning wheels and progress bars should be used to indicate that the application is actually working.

- **Data size experiment**: To back up our decision in choosing JSON instead of XML, we are going to compare the two document sizes. A successful result will show that XML documents are larger than JSON documents, meaning we made the right decision.

6.2 Experiments and results

6.2.1 Acceptance test

Acceptance tests are conducted on the final implementation of the system to determine if the functional requirements are fulfilled. The tests will confirm if the RIS delivers the functionality as promised. The acceptance tests we are conducting are two real-life cases. The cases are listed below.

- **Case 1**: In this test-case we are going to see if the system is able to help the user in finding route information and bus stops based on the user only knowing the address of where he is going. The user will activate the notification service to receive help concerning when to get off the bus. This test-case will use all the front-end use-cases listed in Figure 3.1 and all the back-end use-cases listed in Figure 3.2.

- **Case 2**: The second test-case is similar to the first test-case, but instead of searching for an address we are going to search for a place. This test is conducted to prove the flexibility of the search function.

For each test-case we will list all the back-end use-cases being utilized. We will include a result describing the output from each of the use-cases. Here we will see if the system meets the specified requirements.

In the test-cases we will use the application under real-life circumstances. The user conducting the tests will travel from one bus stop to another. He will act as a stranger and fully rely on the output produced by the application. Based on the information offered by the mobile application the user will find out where the bus stops are located, which bus to take, when the bus departs and arrives to the bus stops, and when he is close to the destination bus stop.

Test-case 1

In this test-case we are located at the University of Tromsø (lat=69.681389, lng=18.976709) and are going to the address Nedre Markveg (lat=69.662849, lng=18.958422). The application is used for finding the bus stops we are traveling between, which in this case are UiT/ISV and Storskogen. Timestamps listing when the

bus departs and arrives are also used for planning when we should start walking to the bus stop. Listing 6 shows the result of our test-case. In the table we refer to the requirements specified in the back-end use-case diagram in Chapter 3.2.

Use-case	**Result**	**Pass/fail**
Find bus stops close to destination defined in search.	The usage of the mobile application started with finding bus stops close to the address. The RIS were able to find 6 bus stops based on the user's current location and input address.	Pass
Find bus stops close to user that corresponds with destination bus stop.	The result of the previous function listed a set of bus stops close to the destination. When the user now selects where he wants to go, in this case a bus stop called Storskogen, the RIS will find bus stops close to the user, based on the destination bus stop and the current location. The result after selecting the arrival bus stop are 11 bus stops close to the user.	Pass
Find timestamps based on two given bus stop identifiers.	When the user selects the bus stop he wants to travel from, in this case UiT/ISV, the parameters needed for extracting timestamps are known. The result of the timestamp extraction are two different routes, 20 and 21, and three departure– and arriving timestamps for each route.	Pass
Find distance and driving time between current location and arrival bus stop.	When the user activates the notification service, the application will start sending requests to the server to find the time and distance remaining of the travel. Based on this data the application is able to know how close the user is to the bus stop.	Pass

Listing 6: Test-case 1: Testing the back-end use-cases with address search.

In Listing 6 we can see that the RIS was able to fulfill each requirement specified in the use-case diagram. A problem concerning the accuracy of the notification service did however occur.

When we were less than 200 meters from the destination bus stop, the mobile device did not vibrate as it should. However, when we got of the bus it started vibrating right away. A list containing issues that could have caused the inaccuracy is listed below. We will also suggest and test different approaches that may solve this problem.

- **GPS**: GPS inaccuracy on the iPhone could have been the reason behind the problem. We noticed that the blue dot indicating our current location only moved every 3-5 seconds. Sending coordinates representing a location we have passed to the RIS, would have caused wrong values being returned to the application. The wrong distance will impact the timing of when the application should make the

phone vibrate.

- **Update frequency**: In 5 seconds a bus is able to drive 80 meters if it keeps 60 km/h. If a request is not initiated when the bus is between the 200 meter radius border and the bus stop, the user will either not receive a notification, or it will be sent when the bus has passed the bus stop. The probability of not receiving a notification at all is small, since the bus must drive 400 meters in 5 seconds. The probability of not receiving a notification when the bus is between the radius and the bus stop, is also minimal since the bus must keep an average speed of 144 km/h. This means that the update frequency is unlikely to cause the inaccuracy problem. However, the 5 second frequency will not give the best accuracy since the notification may be sent when the bus has driven a couple of meters within the radius.
- **Notification radius**: The notification radius determines when a notification should be initiated. In our tests we sat the radius to be 200 meters. With correct coordinates and a low response time from the RIS, a 200 meter radius should have worked fine.
- **Google Maps web-service**: The results RIS receives from the Google Maps web-service may be wrong. It finds the shortest road-path between the user's current location and the destination bus stop. The path used by the web-service to calculate the distance and the time left, may be a path that is not the shortest, or a path the bus is not actually following. The RIS is unable to validate the correctness of the path used in the calculations since its all handled by Google Maps.

With a list of issues that could have caused the inaccuracy, we will suggest two approaches that could fix the problem:

- **Increasing the radius:** The first approach is simply to adjust the radius from 200 to 400 meters. This will create a larger circle around the bus stop, and give the system more time to notice that it is actually inside it.
- **Using air distance:** Calculating the air distance between the user's current location and the bus stop, is the second approach. In this case we can use the data received from the RIS (distance and time left) to update the user-interface, while we use the air distance to determine when the notifications should be sent. The air distance can be calculated locally on the iPhone.

In a last attempt to make the notification service work properly, we ran some experiments using both the before mentioned approaches. We increased the radius from 200 to 400 meters, and used air distance to calculate the distance between our current location and the destination bus stops. The outcome of this experiment was a far more accurate notification service. We drove by different bus stops with the notification service activated, and we always received the notifications in good time before arriving at the bus stop. This result shows that the distance extracted from the Google Maps web-service are in some cases wrong, which may be caused by it returning values concerning a path the bus is not following.

In these experiments we have found that the air distance calculated locally is far more reliable than the distance returned by the Google Maps web-service.

Use-case	**Result**	**Pass/fail**
Find bus stops close to destination defined in search.	The usage of the mobile application for this case started with finding bus stops close to the place we were going to (Nerstranda). The RIS were able to find 7 bus stops based on the user's current location and the defined place.	Pass
Find bus stops close to user that correspond with destination bus stop.	The result of the previous function lists a set of bus stops close to the destination. When the user now selects where he wants to go, in this case a bus stop called Wito (since it is closest to Nerstranda), the RIS will find bus stops close to the user based on the destination bus stop and the current location. The result after selecting the arrival bus stop are 11 bus stops close to the user.	Pass
Find timestamps based on two given bus stop identifiers.	When the user selects the bus stop he wants to travel from, in this case UiT/ISV, the parameters needed for extracting timestamps are known. The result of the timestamp extraction are two different routes, 20 and 21, and three departure– and arriving timestamps for each route.	Pass
Find distance and driving time between current location and arrival bus stop.	When the user activates the notification service, the application will start sending requests to the server to find the time and distance remaining of the travel. Based on this data the application is able to know how close the user is to the bus stop.	Pass

Listing 7: Test-case 2: Testing the back-end use-cases with place search.

Test-case 2

In the second test-case we are located at the University of Tromsø (lat=69.681389, lng=18.976709) and are going to the shopping mall Nerstranda (lat=69.646800, lng=18.954800). In this case we are going to focus on the flexibility of the search function. Instead of searching for an address we are going to search for a place, in this scenario Nerstranda. Listing 7 shows the result of this test-case. In this table we also refer to the requirements specified in Chapter 3.2. The results show that the system's search function is flexible by offering different destination types. The feature will especially be useful for people who are unacquainted with bus stop names.

6.2.2 The user-interface

In Chapter 4.2.2 we listed a sketch illustrating what the user-interface for the application should look like, and in Chapter 5.2.5 we presented screenshots from the prototype application. To evaluate if we were able to implement a user-interface similar to the sketches, we will do a comparison. Since most of the windows in the prototype application conforms with the sketches, we are not going to list all the figures once again in this section. Instead we are going to take a closer look on a window that did not turn out as we planned, namely the notifications window.

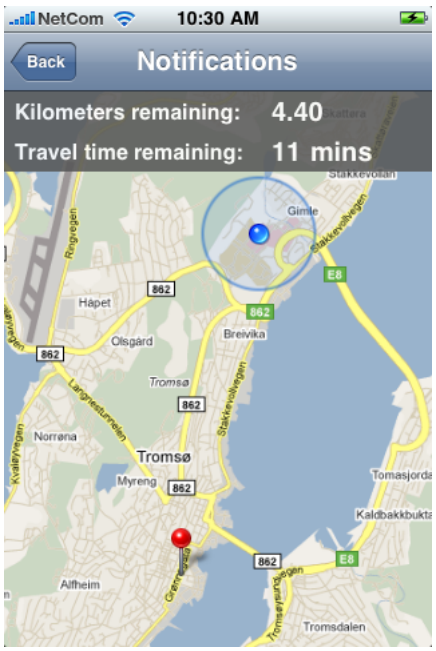

Figure 6.1: Prototype: Notification service.

Figure 6.2: Sketch: Notification service.

The notification service tells the user when he is close to the destination bus stop. When activated, the application communicates with the RIS to find out what is remaining of the bus journey. The resource created by the RIS contains two values, remaining distance and time. The original plan was to show only the remaining distance, and to update it as long as the service was activated (see Figure 6.2). We ended up using a better solution with a map illustrating the user's current location (blue circle), and the destination bus stop's location (red pin). We also included the remaining time and distance in the top of the window (see Figure 6.1). The final solution is much more informative and useful.

6.2.3 Performance experiment

The performance experiments are conducted to measure the time it takes to create the resources offered by the RIS. The experiments have been run locally on an Apple iMac (iMac5,1) computer. The test machine

Method name	Parameters	Max	Min	Avg
busstops.close.to.search	searchType = address str = Nedre Markveg tromso currLat = 69.681686 currLng = 18.976967	3.52 s	2.52 s	**2.86 s**
busstops.nearby	busStopId = 108 currLat = 69.681686 currLng = 18.976967	5.97 s	5.15 s	**5.50 s**
timestamps.from.to	toBusStopId = 108 fromBusStopId = 118	0.48 s	0.18 s	**0.24 s**
distance.between.coordinates	currLat = 69.681686 currLng = 18.976967 destLat = 69.662125 destLng = 18.959925	0.40 s	0.21 s	**0.27 s**
path.between.coordinates	startLat = 69.662125 startLng = 18.959925 stopLat = 69.666100 stopLng = 18.960150	0.28 s	0.21 s	**0.24 s**

Listing 8: API measurements in seconds. 1.0 kilometer radius.

is equipped with an Intel Core 2 Duo 2.16 GHz processor and 2 GB of memory, and is connected to a 100Mbits network. In this experiment we will use real-life parameters as input for each method in the API. The parameters are the same that were used in the first acceptance test (Section 6.2.1). We will measure each method in the API 10 times, and calculate the average time. The results of the experiments are listed in Listing 8. The unit of measurement is in seconds.

The results from the performance experiments show that the generation time for each resource varies. The results for the last three resources are good, since they come below the 1 second limit. The resources busstop.close.to.search and busstops.nearby are not created as fast as they should, but still, they are below the 10 second limit. Since the generation time is so long, it makes the user experience in the mobile application less seamless.

We have tried to address the parts of the source code that make the system slow by doing internal measurements[2]. The result of these measurements indicates that we have two bottlenecks in our system. The bottlenecks are as follows.

- **Google Maps API/Parameters web-service**: The Google Maps web-services are used for finding road distance and walking time between two coordinates, and converting addresses to coordinates. We

[2]We have measured the time it takes to run the different functions in our source code.

Method name	Parameters	Max	Min	Avg
busstops.close.to.search	searchType = address str = Nedre Markveg tromso currLat = 69.681686 currLng = 18.976967	1.74 s	1.04 s	**1.26 s**
busstops.nearby	busStopId = 108 currLat = 69.681686 currLng = 18.976967	2.06 s	1.48 s	**1.65 s**

Listing 9: API measurements in seconds. 0.5 kilometer radius.

have found that these calls are quite expensive. Our measurements indicate that it takes approximately 200ms calling the web-service and getting the result back to the RIS. In one of our functions we call the Google Maps web-service to find the road distance and walking time for a set of bus stops. In the acceptance test, 6 bus stops were found near the address we searched for. This means that it takes approximately 1.2 seconds for the RIS to find the road distance and walking time for all these bus stops. And if more bus stops are found, it will take even longer. Calculating the air distance locally instead of getting the road distance from the Google Maps API/Parameters web-service could have solved this issue. But this would have given far more inaccurate information to the user. The user may accept the trade-off our system implements if he tolerates the extra time the system uses to get the accurate information.

- **Database extraction**: The second bottleneck is a function that extracts route information from the database. The function receives two arrays, one containing bus stops close to the user, and one containing bus stops close to the selected destination. It is responsible for finding all possible travel combinations based on these two arrays. This means that the number of elements inside the arrays determines how many possibilities have to be checked. In the first acceptance test, 272 database queries were executed to find possible travel combinations. To minimize the number of queries we have to decrease the number of bus stops in the input arrays. In the results shown in Listing 8, we used 1.0 kilometer as the radius whem finding bus stops nearby and close to the destination. If we decrease the radius to 0.5 kilometer the number of bus stops will decrease, meaning less queries. In Listing 9 we conducted the same experiments, but decreasing the radius to 0.5 kilometers [3]. The result shows that the resources, busstop.close.to.search and busstops.nearby, are built much faster. In this case there were only executed 30 queries. Reducing the radius means a quicker system, but also less travel combinations. The radius reduction also cuts down the number of requests being sent to the Google Maps web-service, which also improves the system's response time.

[3]Since the radius decrease does not expose the resources, timestamps.from.to, distance.between.coordinates, and path.between.coordinates we do not list them in Listing 9.

6.2.4 JSON vs XML

In our system we choose to use JSON as our data-interchange format. The background for this decision is the overhead the XML tags would have produced in our system's data transfers. To prove wether we were right or wrong, we took the output produced by busstops.close.to.search and compared the two formats. The JSON output used in this experiment is taken from the acceptance test in Chapter 6.2.1. The result of the comparison is listed in Listing 10.

Format	**Size (bytes)**
JSON	664
XML	746

Listing 10: JSON and XML size comparison. Smaller is better.

The result of the comparison backs up our theory based on JSON being lighter than XML. In our experiment the JSON resource turned out to be 82 bytes smaller than the XML resource. The XML resource was built as small as possible. The keys and values in the JSON resource were mapped straight into attributes, making the XML document as small as possible. The difference being so small will not have a major impact on the performance of our system. However, if the user downloads the resource 13 times, he would have transferred 1 MB less with JSON compared to if we had used XML. This means that over time the user will save data-traffic with JSON.

6.2.5 Non-functional requirements

Some of the non-functional requirements can not be measured. These requirements are usability, accessibility, platform independence, and the interface. We will instead discuss how our implementation handles these requirements.

Usability

The usability requirement focuses on creating a logic, simple, and clean mobile application. In our design we start asking the user *where he is going*, instead of asking *from where he is going*. This makes the application much more usable because it does not require the user to know where he is departing from. By following this approach, our application is usable for people knowing where the bus stops are located, and for those who do not know where they are located. This means that more people will be able to use the application. To improve the usability even more, we have made it possible for the user to search for different destination types (address, bus stops, and points of interest) as well.

Accessibility

This requirement focuses on being capable of accessing route information from any location. The only solution to fulfill this requirement is by placing the RIS "on the Internet". This will make the RIS accessible for clients located anywhere. When building and testing the system, we deployed the RIS on the Internet giving it a unique IP address, i.e. we installed the RIS on a web-server. In all the test-cases conducted, we experienced no issues communicating with the RIS.

Platform independence

An important requirement our system fulfills, is platform independence. The main logic in our system is deployed in the RIS. The RIS offer its functionality through a RESTful web-service. The requirements to use the web-service are; support for HTTP, and being able to parse JSON documents. Most mobile platforms today have libraries which lets the application send HTTP requests. JSON libraries are also well supported on mobile platforms because of its small overhead.

RIS interface

The functionality provided by the RIS is offered through a simple web-service containing 5 functions. Each function offers the client a resource containing information based on the parameters it receives. Our goal has been to design an interface that decouples the functionality of the system as much as possible. For instance, the resource busstops.nearby could contain route information for all the bus stops that were found nearby. But instead of adding them to this resource, we put them in the resource called timestamps.from.to, which is responsible for supplying timestamps based on two given bus stops.

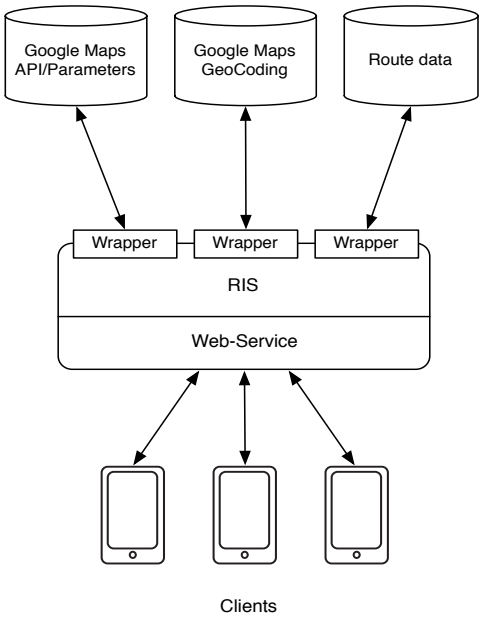

Figure 6.3: A high level illustration of our architecture.

6.3 PIA context

The RIS implements a wrapper for each resource it extracts data from. They collect only the data that is needed by the RIS. A high-level figure that illustrates this architecture is shown in Figure 6.3. Since we know the resource requirements of the RIS, we can suggest how a personal cloud can make things easier. In Figure 6.4 we show a new architecture containing a personal cloud.

6.3.1 Decomposition

In this section we will describe how the architecture containing the personal cloud is decomposed (see Figure 6.4).

The clients (1) will not have to be re-implemented. They will use the same interface offered by the RIS (2). The RIS will not communicate directly with the resources (5) in this architecture. Instead, it will utilize a simple bus API (3) offered by the personal cloud (4). The API is placed between the resources and the RIS. This is easier and less time consuming to use, compared to implementing wrappers for each resource. The bus API will receive data from the same resources (or maybe more) as the system we have presented in this thesis. It must provide a set of functions the RIS can use to fulfill its requirements.

The motivation behind the personal cloud is to gather personal information and offer it through APIs. The APIs can then be used to create *personal services*[4]. A couple of other master students are also working

[4]A personal service is a service that utilizes one or more API's provided by the personal cloud. E.g. the RIS using the bus

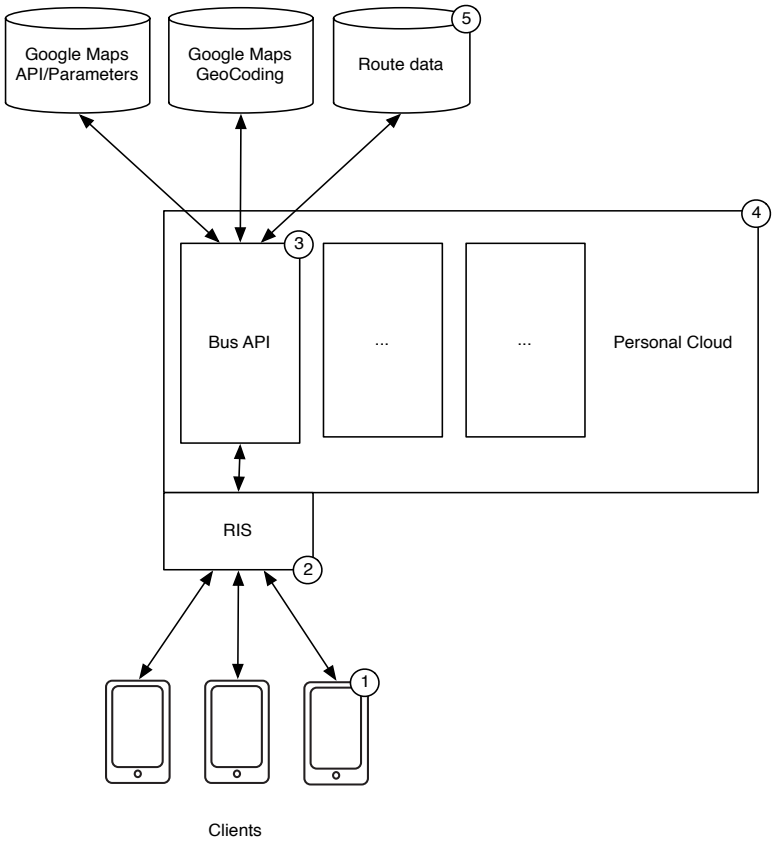

Figure 6.4: Personal cloud architecture.

on the PIA project, and we have found an interesting example of a personal service that can utilize the bus API and a shopping API.

A shopping system have been designed and implemented to let users scan barcodes of a product and being able to see price differences based on different stores. Location is used to pinpoint nearby stores that offer the scanned product. A problem the user faces is how he should get to the store. If the shopping system utilized the bus API, it could have provided additional information suggesting what bus the user should take to get to the store. A public service offering this functionality could simply have been built by using both these APIs.

6.3.2 The Bus API

Listing 11 contains a set of functions the bus API should provide. The functions are suggested based on the resource requirements specified in Chapter 3.4.

API.

Method	Parameters	Return value
Convert address to coordinates (GeoCoding).	address (String)	Returns the longitude and the latitude of the input address.
Find bus stops close to a location.	latitude (Float) longitude (Float) radius (Float)	Returns bus stops that are close to a location. The radius specifies the extent of the search. Distance and walking time will also be returned.
Check if two bus stops exist together in any routes. Also check if it is possible to travel from A to B.	fromBusStop (Integer) toBusStop (Integer)	Returns routes where both occur and where it is possible to go from A to B.
Find timestamps based on two bus stops.	fromBusStop (Integer) toBusStop (Integer)	Returns routes and the next three timestamps.
Find road distance and driving time based on two coordinates.	currLat(Float) currLng(Float) destLat(Float) destLng(Float)	Returns two values, distance and time it takes to drive between the two locations.
Path between two coordinates.	startLat(Float) startLng(Float) stopLat(Float) stopLng(Float)	Returns a set of coordinates that can be used to draw a graph in a map.

Listing 11: The Bus API.

6.4 Summary

In this chapter we have evaluated and tested our system. The experiments, tests, and measurements were first addressed in Section 6.1. An acceptance test was then conducted in Section 6.2.1 to determine wether the system fulfilled the functional requirements or not. Two test-cases were executed and both were successful. The notification service in the application did not perform as we had hoped during these tests. We experienced inaccuracy which resulted in the notification service not working as planned. Further testing found that the values returned from the Google Maps API/Parameters web-service were wrong. To solve this issue we used a different approach when calculating the distance between the user's location and the destination bus stop. A comparison of the mobile application and the sketches were done in Section 6.2.2 to see if our design goals were fulfilled. The result showed that the notification service ended up being far more informative than we first planned. A performance experiment was conducted in Section 6.2.3 to measure the time it takes the RIS to build the resources it offers. The experiment resulted in the discovery of two bottlenecks that increased the build time. It also made us aware that the bigger search radius we use, the slower the system will behave. The last experiment was conducted in Section 6.2.4 to determine the size

difference between JSON and XML. JSON turned out to be smaller than XML as we initially thought. In Section 6.3 we discussed how a personal cloud could make things easier. We also introduced a new architecture illustrating how services can be created based on extracting personal information through APIs in the personal cloud. An API our system would require from the personal cloud was also addressed. We suggested some functions it should have provided as well.

Chapter 7

Conclusion

In Chapter 1 we introduced how timetables and route information are distributed by bus companies today, and the drawbacks of these methods. Systems have been built to ease the task of doing lookups in printed timetables and PDF documents. However, these systems are not context-aware. They lack the ability to map the user's current location and travel interests into the application. We propose a system that is able to find bus stops and departure times based on a single user input, the destination. Bus stops are suggested by the application based on the destination specified by the user, and the user's current location. The system consists of two main components, a RIS (Route Information Server) and a mobile application.

PIA is a project that focuses on making personal information more available, a project introduced in Chapter 2. PIA suggests that a personal cloud or a middleware platform is built to integrate and share personal information. The goal of the system is to increase the PIM, i.e making it easier to find information you are interested in, at a given time. The problem with most of the systems mentioned in Chapter 2 is that they focus on local PIM. SEMEX, Haystack and Semantic Desktop are concepts that focus on making it easier to find information on a local computer. The personal cloud introduced in PIA, is a system that increases the PIM not locally, but in a distributed fashion.

The requirements for the system was listed in Chapter 3. Requirements concerning what resources the RIS needs are important. Without the resources it is unable to implement and fulfill its requirements. A design describing how the requirements are fulfilled was suggested in Chapter 4. The design contains an architecture with a description of how the components are tied together. In Chapter 5 we described some implementation details.

An evaluation of the system was given in Chapter 6, where real-life acceptance tests and performance experiments were conducted and described. Thoughts concerning how a personal cloud could be designed to provide personal information was described in this chapter as well.

7.1 Achievements

The problem definition we defined in Chapter 1.2 is stated below:

> *Our goal in this thesis will be to design and implement a system that offers route information in a ubiquitous and simple way. A system will be built offering a set of resources required by a context-aware mobile bus application. The bus application will also be implemented for testing and evaluating the system's functionality. The user's context will play an important role when extracting information. The system will offer its functionality through a web-service as a set of resources. The resources will contain data extracted from a variety of location– and route information services. Our objective is also to address the resource requirements of our system, and to find out what functionality the PIA platform must provide if we were to integrate our system in a personal cloud.*

We have developed a system we call RIS that is able to extract and create resources based on a set of context parameters. The interface provided by the RIS have been designed based on the requirements of a mobile context-aware bus application. The RIS have been evaluated in a scientific context, and was found to fulfill the requirements of the mobile application.

In Chapter 6.2 we evaluated the RIS using the mobile application developed for the iPhone. We show that it is able to extract and create resources the application can use for providing route information for the user. The application has been tested under real-life circumstances during the evaluation, and has proved to give useful information to users only knowing the destination of where they are traveling. The system has also demonstrated that it is flexible since it is able to find bus stops close to addresses, places, and buildings. Our system uses a new and unique way of finding route information, by only requesting one parameter from the user, the destination. This parameter has shown to be enough for finding bus stops close to the destination and the user. With this approach we are able to increase the user-experience significantly compared to other systems.

In conclusion, this thesis has presented a system that provides route information in a new and easy way by simply utilizing the user's context. We have also suggested a new architecture illustrating how a personal cloud can provide personal information through different APIs.

7.2 Future work

PIA is a project in its early stage at the University of Tromsø. The work conducted in this thesis is part of the initial phase of the project. We have suggested how a personal cloud could provide personal information through different APIs, and how we are able to create new services by integrating information. Still, the project has several challenges ahead that need to be solved. A more flexible middleware system (personal cloud) that integrates a range of different multimedia types must be designed and implemented. Standards

for how data should be delivered to the middleware should also be specified. Security is another important topic in the project since its focus is targeted at making personal information more accessible.

In future work we will also focus on adding new features to the system we have built and presented in this thesis. The first feature would be to add support for transitions. For instance, if the user has to change bus during the travel to get to a specified destination, the system should be able to present the different routes and bus stops the user must travel by. The current system is only able to find travel alternatives containing two bus stops, a departing bus stop and an arrival bus stop. The second feature would be to replace the old route information source with a new one, containing more information. Our database today, contains only three different routes for Tromsø. In a future system we would expand and use all the routes for entire Troms county. The third and last feature we are suggesting, is to implement real-time geo-tracking of buses. This feature can be implemented with the help of users that have activated the notification service in the mobile application. Since they frequently send requests to the server containing their current location while traveling, we can simply connect the coordinates to the trip number. This feature would make it possible for users to see where the bus is located in almost real-time. It also makes it possible to estimate how much the bus is delayed, and when it will arrive at the bus stop the user is located at.

We will also focus on making the RIS more responsive. The results of the experiments showed that some of the resources have improvement potential when it comes to performance. The application's notification service should also become more functional. Instead of indicating when the user is getting close to a bus stop, it should notify when the user should press the stop button.

Bibliography

[1] E. Adar, D. Karger, and L. A. Stein. Haystack: per-user information environments. In *CIKM '99: Proceedings of the eighth international conference on Information and knowledge management*, pages 413–422, New York, NY, USA, 1999. ACM.

[2] A. Andersen, R. Karlsen, and G. Blair. Project description - pia: Personal information and assets. Tromsø, Norway, November 2009.

[3] O. Bergman, R. Boardman, J. Gwizdka, and W. Jones. Personal information management. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1598–1599, New York, NY, USA, 2004. ACM.

[4] T. Bray, J. Paoli, E. Maler, F. Yergeau, and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.0 (fifth edition). W3C recommendation, W3C, Nov. 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[5] P. J. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young. Computing as a discipline. *Computer*, 22(2):63–70, 1989.

[6] X. Dong and A. Y. Halevy. A platform for personal information management and integration. In *CIDR*, pages 119–130, 2005.

[7] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 72–79, New York, NY, USA, 2003. ACM.

[8] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. Chair-Taylor, Richard N.

[9] D. Huynh, D. R. Karger, D. Quan, and V. Sinha. Haystack: a platform for creating, organizing and visualizing semistructured information. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 323–323, New York, NY, USA, 2003. ACM.

[10] J. Jarvinen, J. DeSalas, and J. LaMance. Assisted gps: A low-infrastructure approach. *GPS World*, 2002.

[11] W. Jones. How is information personal? In *Personal Information Management: PIM 2008, CHI 2008 Workshop*, April 2008.

[12] J. Nielsen. *Usability Engineering.* Morgan Kaufmann Publishers, San Francisco, California, October 1994.

[13] L. Sauermann, A. Bernardi, and A. Dengel. Overview and outlook on the semantic desktop. In Dennis and L. Sauermann, editors, *Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference*, 2005.

[14] J. Teevan and W. Jones. The disappearing desktop: pim 2008. In *CHI '08: CHI '08 extended abstracts on Human factors in computing systems*, pages 3917–3920, New York, NY, USA, 2008. ACM.

Appendix A: Tools

In this appendix I have included a list of tools that have been used throughout this project.

LaTeX

- **LaTeX:** This thesis is written in LaTeX. LaTeX is a document markup language[1].
- **Code listings:** Code is listed with the help of two packages. *Pygmentize*, a syntax highlighter written in Python[2]. A latex package that is called *minted*[3] have been used to format source code inside the .tex files. Minted requires pygmentize to work.
- **Figures:** The figures included in this thesis are mostly created by an application called OmniGraffle [4]. Some figures are created with the help of Adobe Illustrator [5]. All figures in this thesis are vector figures except the iPhone screenshots.

Development tools

- **Mac:** All work on this thesis have been conducted on an Apple iMac running OS X 10.6. Most of the IDE's I have used are Mac-only applications.
- **Thesis writing:** The LaTeX source have been written in an IDE called TextMate [6].
- **Development:** The RIS have been developed in PHP, the IDE used for this is called Coda [7]. The iPhone application was written in Xcode [8]. The Python scripts I used for benchmarking the system and for parsing the route information was written in TextMate.

[1] LaTeX: http://en.wikipedia.org/wiki/LaTeX
[2] Pygmentize: http://pygments.org/download/
[3] Minted: http://code.google.com/p/minted/
[4] OmniGraffle: http://www.omnigroup.com/products/omnigraffle/
[5] Illustrator: http://www.adobe.com/products/illustrator/
[6] TextMate: http://macromates.com/
[7] Coda: http://www.panic.com/coda/
[8] Xcode: http://developer.apple.com/technologies/tools/xcode.html