



NTNU – Trondheim
Norwegian University of
Science and Technology

TransitVision: Approximating Vehicle Locations Using SIRI-SM Real-Time Data

Sofia Nascimento Bakke
Ole Kristian Nakken

Master of Science in Computer Science

Submission date: June 2015

Supervisor: Rune Sætre, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Many public transportation agencies around the world track their vehicles using GPS. However, the GPS data is usually not directly accessible by passengers, but is used to provide real-time arrival estimates. This thesis attempts to approximate the locations of vehicles using the available real-time arrival estimates, through a smartphone application named TransitVision. By utilizing the SIRI standard¹, TransitVision is interoperable with other public transportation agencies.

A preliminary study on travelers' habits and perceived waiting time, created a solid foundation for TransitVision. This thesis also includes a study into the state-of-the-art, to examine other applications, technologies and discover limitations. The study included a thorough examination of several SIRI implementations, to consider how to incorporate them in TransitVision. Finally, 15 testers evaluated the application's usability through the System Usability Scale².

The preliminary study indicated that mobile transit application users believed they waited on average five minutes, which was about the same as those using non-digital retrieval methods. TransitVision was initially developed for Oslo, but was also tested on transit data from Tampere. It worked well in both cities, confirming TransitVision's interoperability. However, other cities require major modifications to TransitVision. TransitVision achieved a System Usability Scale score of 86, which is close to a superior score.

¹Service Interface for Real Time Information, more at www.siri.org.uk

²More information on SUS at <http://www.measuringu.com/sus.php>





Sammendrag

Flere kollektivtransportsselskaper verden over har systemer for å spore kjøretøyene sine med GPS. GPS-dataen er som oftest ikke direkte tilgjengelig for passasjerer, men blir brukt til å oppgi sanntidsestimater for ankomst. Denne masteroppgaven prøver å tilnærme en plassering av kjøretøy ved å bruke tilgjengelig sanntidsestimater. Dette blir gjort med en smarttelefonapplikasjon som heter TransitVision. Ved å utnytte SIRI-standarden³ kan TransitVision modifiseres til å fungere med andre kollektivtransportsselskaper.

En forstudie om reisevaner og oppfattet ventetid dannet et solid grunnlag for TransitVision. Denne masteroppgaven inkluderer også en “state-of-the-art” studie for å undersøke andre applikasjoner, teknologier og oppdage begrensninger. Studien inkluderte en grundig undersøkelse av flere SIRI-implementasjoner for å se hva som kreves for å innarbeide de i TransitVision. Til slutt evaluerte 15 testere applikasjonens brukbarhet ved hjelp av System Usability Scale⁴.

Forstudiet indikerte at reisende som brukte mobilapplikasjoner for å hente sanntidsinformasjon om kollektivtransporten tror de venter 5 minutter i gjennomsnitt, som var omtrent det samme som de som brukte ikke-digitale metoder. TransitVision var i utgangspunktet utviklet for Oslo, men ble også testet på transportdata fra Tampere. Det fungerte bra i begge byene, noe som bekrefter TransitVision sin interoperabilitet. Allikevel kan noen andre byer kreve større modifikasjoner av TransitVision. TransitVision oppnådde en System Usability Scale score på 86, noe som er nære en overlegen poengsum.

³Service Interface for Real Time Information, mer på www.siri.org.uk

⁴Mer informasjon om SUS på <http://www.measuringu.com/sus.php>





Preface

This report presents the master project conducted by Sofia Nascimento Bakke and Ole Kristian Nakken. The work is a fulfillment of a Master of Science in Computer Science at the Department of Computer and Information Science at the Norwegian University of Science and Technology. The project was supervised by Associate Professor Rune Sætre.

The thesis is a contribution to the FUIROS project. While earlier FUIROS projects have primarily focused on the use of natural language for route querying in Trondheim, this thesis focuses on displaying real-time public transit data in Oslo on a smartphone map application.





Acknowledgements

We would like to thank our supervisor Rune Sætre, for his guidance throughout the development of this master thesis. We would also like to thank Carl-Fredrik Sørensen, for helping with proofreading and structuring the report. Thank you to all the people participating in both the preliminary study and the usability test of TransitVision. Finally, we would like to thank our friends and family for their moral support throughout this project.





Problem Description

An increasing number of public transportation companies around the world now provide real-time estimates of bus arrival times. Most of them also give developers open access to these estimates via web APIs. However, the majority of these companies restrict access to the actual GPS coordinates. A few cities like Tampere, Finland, and New York, USA have successfully given GPS access to software developers. These cities now experience a great influx of real-time applications, for both smartphone and web, created by third parties.


Because GPS coordinates are still not available in Norway, this project aims to use the estimated arrival times to approximate the locations of vehicles in transit. To ensure portability, the SIRI standard should be used.

The application will be tested on at least 12 real users to measure its usability and whether the implementation is a satisfactory representation of real-time data. The test will include a scenario test where the application's features are tested, followed by a System Usability Scale questionnaire.





Contents

Abstract	i
Sammendrag	iii
Preface	v
Acknowledgements	vii
Problem Description	ix
List of Tables	xv
List of Figures	xvii
List of Listings	xix
1 Introduction	1
1.1 Background and Motivation	1
1.2 Goals and Research Questions	2
1.3 Research Method	2
1.4 Thesis Structure	4
2 Theory and Background	5
2.1 BusTUC and FUIROS	5
2.2 Digital Schedule Formats	6
2.2.1 The Norwegian REGTOPP Standard	6
2.2.2 The GTFS Standard	6
2.2.3 NeTEx	7
2.3 Real-time Vehicle Tracking Formats	7
2.3.1 SIRI	8
2.3.2 NextBus	11
	xi

2.3.3	GTFS-Realtime	11
2.4	Google Maps	12
2.4.1	The Google Maps API	13
2.4.2	The Google Directions API	13
2.5	Server Technologies	14
2.5.1	Node.js	14
2.5.2	MongoDB	15
2.6	A Brief Look at Similar Applications	15
2.6.1	Google Maps	16
2.6.2	RuterReise	16
2.6.3	Bartebuss	16
2.6.4	Nettbuss	16
2.6.5	Busskartet.no	17
2.6.6	Tampere Bus Map	17
2.7	The System Usability Scale	18
2.7.1	Calculating the SUS Score	18
3	Research and Development Method	21
3.1	Preliminary Survey	21
3.2	Finding and Prioritizing Requirements	22
3.3	Designing TransitVision	23
3.4	Testing Server Compatibility	23
3.5	User Test	24
4	Results	27
4.1	Results from the Preliminary Survey	27
4.1.1	Waiting Time	27
4.1.2	Application Usage Feedback	29
4.2	TransitVision Requirements	31
4.2.1	Quality Attributes	31
4.3	The Technologies Behind TransitVision	33
4.3.1	Targeting a Platform	33
4.3.2	Selecting a Server Framework	34
4.3.3	Selection of Database Technology	34
4.3.4	Map APIs	35
4.4	System Architecture	35
4.4.1	Architectural Pattern	36
4.4.2	Class Structure of Mobile Application	36
4.5	Design	38
4.5.1	Estimating Positions	39
4.5.2	Mock-Ups	39



4.5.3	Line Search Tools	41
4.5.4	Marker Icons	42
4.5.5	Flow Chart	43
4.6	Implementation of TransitVision	44
4.6.1	Mobile Application	44
4.6.2	Providing Vehicle Positions	46
4.7	SIRI Interoperability	50
4.7.1	Server Limitations	50
4.7.2	Prerequisites for the SIRI-SM Implementations	52
4.7.3	Prerequisites for the SIRI-VM Implementations	52
4.7.4	Compatibility of Different SIRI Implementations	53
4.7.5	Expanding to Tampere	54
4.7.6	Prospect of including Kolumbus' API	54
4.8	User Testing	55
4.8.1	Feedback Received from User Testing	55
4.8.2	The SUS Score	56
5	Discussion	59
5.1	Preliminary Study	60
5.2	Fulfillment of Requirements	60
5.3	Obstacles Found During Development	61
5.3.1	Vehicle Animations	61
5.3.2	Irregularities in Ruter's Data	61
5.3.3	Google Directions API	62
5.4	User Test	63
5.5	Research Questions and Goal Achievements	63
5.5.1	The Goal	64
5.5.2	Research Question 1	64
5.5.3	Research Question 2	64
5.5.4	Research Question 3	64
6	Conclusions	65
6.1	Future Work	66
6.1.1	Connecting the Oracle with the Map	66
6.1.2	Expand to Other Cities	66
	Acronyms	69
	Bibliography	71



A	SUS	75
A.1	The Questionnaire	75
A.2	Error Bar Diagram	76
B	SIRI-VM Example	77
C	Digital Attachements	79
C.1	Github Repositories	79
C.2	TransitVision Google Play Link	80
C.3	Video of TransitVision in Action	81
D	Setting Up the Project	83
E	Preliminary Study	85
E.1	Preliminary Study Questionnaire	85
E.2	Waiting Time Survey Answers	86



List of Tables

1.1	Project Milestones	3
4.1	Reasons for Being Too Late for the Bus	28
4.2	Average Percieved Waiting Time by Route Retrieval Method	29
4.3	Average Frustration by Route Retrieval Method	29
4.4	TransitVision Requirements	32
4.6	Stop Visit Requests from Multiple Machines to Ruter's API	51
4.7	Network Traffic Limitations for Kolumbus' API	51
4.8	Comparison of SIRI Implementations	53
4.10	SUS Results	56
4.11	Statement Statistics	57
5.1	Status of Requirements	60
E.1	Introductory and Bus Usage Questions	86
E.2	Waiting on the Bus	90
E.3	App Questions	95





List of Figures

2.1	Google Transit in Oslo	6
2.2	Results of APTA Survey: Arrival Time Formats (Grisby, 2013)	7
2.3	Google Maps Real-Time Arrivals in Boston	12
2.4	Public Transit Applications	15
2.5	Busskartet Displaying Buses around Campus Gløshaugen	17
2.6	Tampere Public Transport’s Traffic Monitor	18
4.1	Perceived Waiting Time	28
4.2	Frustration Level for Public Transit in Trondheim	30
4.3	Popularity of Mobile Transit Applications	30
4.4	TransitVision Architecture	36
4.5	Final Class Diagram for the Mobile Application	37
4.6	Interactions Between User, API, Database and Ruter	38
4.7	Navigation Drawer Menu, Tabbed Menu and Spinner Menu	40
4.8	Mockup Screens	41
4.9	Collapsed SearchView	41
4.10	The Search Components	42
4.11	Early Icons	42
4.12	Final Icons	43
4.13	Flow Chart of TransitVision	44
4.14	The Final Map Screen	45
4.15	Oracle Example	46
4.16	Position Estimation Model	49
4.17	TransitVision using Tampere SIRI-VM Data	54
A.1	SUS Answers With 95 % Confidence Level	76
C.1	Links to Public Code Repositories	79
C.2	Link TransitVision in Google Play Marketplace	80
C.3	Link to Video Showing TransitVision in Action	81





List of Listings

1	A Stop Visit Returned from Ruter's API	10
2	NextBus Vehicle Location on Line 2 of the Los Angeles Metro . . .	11
3	Excerpt of MBTA GTFS-realtime response	12
4	One of the Legs on a Route Through Oslo	14
5	Response From OsloTUC	47
6	Position of a Subway on Ruter's Line 1	50
7	SIRI-VM JSON Returned from Tampere Bus Service Online API .	77





Chapter 1

Introduction

TransitVision is an Android application capable of approximating locations of public transit vehicles, using real-time arrival estimates. As many public transit agencies deny direct access to Global Positioning System (GPS) coordinates, TransitVision serves as an alternative method for finding vehicle locations. It focuses on the area in and around Oslo, Norway, but TransitVision is interoperable to many similar systems.

This thesis includes a preliminary study on schedule retrieval habits, and perceived waiting time of travelers. After completing TransitVision, 15 participants took part in a usability test, to measure how well such an application can convey real-time transit information.

The background and motivation of this thesis, and the development of TransitVision, are summarized within this chapter. It includes the goal of the project, and the three research questions guiding the development. Additionally, it contains a section about the research method, which defines the approach of the project. Finally, the thesis structure is outlined.

1.1 Background and Motivation

Bus, The Understanding Computer (BusTUC), or the Futures Ultimate Intelligent Route-Organizing System (FUIROS), is a collection of projects consisting of several master theses from the Norwegian University of Science and Technology (NTNU). BusTUC is a service, allowing natural queries on public transit schedules in Trondheim.

Many Norwegian public transit agencies have implemented real-time tracking systems of the vehicles. However, they restrict direct access to these, and instead provide real-time arrival time estimates at individual stops. In cities



like Tampere, Finland, and New York, USA, agencies have successfully deployed solutions, which present passengers with locations of public transit vehicles. A Norwegian example of an online bus map is `Busskartet.no` (Section 2.6.5), which uses timetable data to visualize the planned journey of all buses in Trondheim. Combining Busskartet with real-time data could increase its usefulness.

Service Interface for Real Time Information (SIRI) is a modern European Committee for Standardization (CEN) standard for relaying real-time information (CEN/TC278/WG3/SG7, 2005). AtB has not yet deployed SIRI in Trondheim, but has scheduled its release to late 2015¹. Oslo became the primary target for TransitVision, as Ruter has developed a SIRI implementation there. Making an application that can be applied to any SIRI system would certainly increase the number of potential users.

1.2 Goals and Research Questions

A software goal together with a set of research questions were outlined to guide the direction of this master thesis. Goal fulfillment and answers to the research questions are discussed in Section 5.5.

The Goal Develop a smartphone map application, which approximates the locations of public transit vehicles using the SIRI Stop Monitoring Service (SIRI-SM).

The application will primarily be used as a platform for testing the prospect of such an application, and its usability. After the development of TransitVision, it will be used to help answer the following research questions:

Research Question 1 How can the SIRI standard be utilized to develop an application that can be applied to any SIRI implementation, without considerable modifications?

Research Question 2 How can a mobile map application, using SIRI-SM, provide a satisfactory representation of real-time data?

Research Question 3 What impact does mobile transit applications have on travellers frustration and waiting time?

1.3 Research Method

At the beginning of the project, milestones were outlined to ensure the development would fit within the time constraints. These milestones are displayed in Table 1.1, and they were all finished on time.

¹Personal correspondence with Morten Wæraas, Senior ICT Administrator, AtB, 2015-01-13



Table 1.1: Project Milestones

Milestone	Finished by
Preliminary and State-of-the-art Studies	2014-12-17
Survey Planning	2015-01-25
Application Planning	2015-02-08
Application Development	2015-03-29
User Testing and Survey Execution	2015-05-03
First Draft of Project Report	2015-05-31

Preliminary Waiting Time Study Before the development of the application, 55 travelers participated in a survey containing 15 questions, conducted during the autumn project. The survey included questions about mobile transit application usage, frustration, and waiting time. It serves as a foundation for the rest of the thesis together with the state-of-the-art study. More about how this study was conducted can be found in Section 3.1.

State-of-the-Art Study It was primarily done by using Google Scholar², and other databases from the university library website, such as Oria³ and Scopus⁴. Most of the time went into finding documents related to usability testing, real-time public transit systems, and different standards for communicating real-time transit data. Additional research went into finding applications similar to TransitVision, to map their strengths and weaknesses. The findings are described in Chapter 2.

Survey Planning How to answer the research questions was decided before how to develop the application. In this way, TransitVision would be better suited to answer the research questions. Details about how the survey was conducted can be found in Section 3.5.

Application Planning The basic features of the application were decided during this stage, prioritized into a product backlog. This stage also included selection of architecture, platforms, and technologies. Methods used are described in Chapter 3.

Application Development By following the architecture, the application consists of two parts: a back-end server and a front-end application. Both of

²scholar.google.no

³www.oria.no

⁴www.scopus.com



these were successfully developed in this stage. This was the most time consuming stage, and the results are described in detail in Section 4.6.

User Testing and Survey Execution After finishing the application, it was tested on travellers to answer the research questions. The results from the survey is presented in Section 4.8.

First Draft of Project Report Even though the report was written continuously throughout the spring semester, time was set to focus on the report towards the end. The first draft of the report was to be finished at the end of May, which gave two weeks of polishing before the due date.

1.4 Thesis Structure

This section describes what the different chapters contain and shows the structure of the thesis.

Chapter 1: Introduction presents the goal and research questions, together with the motivation behind them.

Chapter 2: Theory and Background contains the information gathered in the state-of-the-art study. It describes similar transit applications and the technologies behind them.

Chapter 3: Research and Development Method describes how the methods for answering the research questions.

Chapter 4: Results describes in detail how TransitVision works and its interoperability with other SIRI systems. The chapter also contains results from both the preliminary study and the user test.

Chapter 5: Discussion summarizes the results before answering the research questions.

Chapter 6: Conclusions concludes the thesis, including a description of some of the enhancements that can be done to TransitVision. Because TransitVision is a prototype, it needs some improvements before it can be released to the public.



Chapter 2

Theory and Background

The state-of-the-art study examined different real-time standards, technologies, and transit applications similar to TransitVision. Findings from the state-of-the-art study, are described within this chapter.

2.1 BusTUC and FUIROS

This master thesis is part of the BusTUC project, also known as the FUIROS project. Tore Amble developed BusTUC, and released it on the Web in 1996 (Amble, 2000). It is an application that supports written natural-language interaction for bus schedule retrieval. This means that users can ask about schedules in plain text, and get the result printed to them in complete sentences. BusTUC is available at NTNU's website¹, and supports both Norwegian and English queries. AtB, the public transit agency in Trondheim (Team Trafikk until 2010), added BusTUC, or the Bus Oracle as they call it, to their website² in 1999 (Amble, 2000), and it has been there since.

After Amble's initial work, other contributors have made improvements, including many master theses. Bratseth (1997) created an English version of BusTUC, and TaleTUC was a master thesis by Andersstuen and Marcussen (2012), which incorporated speech recognition. Additionally, mobile applications have been developed, giving easy access to BusTUC. One was Tore Amble Buss (TABuss) by Marcussen and Eliassen (2011), incorporating TaleTUC. Another was Multiple-platform approach to the Ultimate Bus Route Information System (MultiBRIS) by Andersstuen and Engell (2011). Other improvements include

¹<http://busstuc.idi.ntnu.no/>

²<https://www.atb.no/spoer-bussorakelet/category1160.html#oracle>



extending the functionality to work in Oslo (Jacobsson, 2015), and incorporating train timetables into the system.

2.2 Digital Schedule Formats

A myriad of different public transit information systems has spread across Europe, as many agencies developed individual standards. A result of this is a lack of interoperability with other modes of transport and nations (Tibaut et al., 2012). This section looks at the most prominent formats for relaying schedule information.

2.2.1 The Norwegian REGTOPP Standard

Regional Trafikkopplysing (REGTOPP) was developed in Norway during the 1990s. It was a result of the growing need for a national standard for relaying timetable information (Oslo og Akershus Trafikkservice AS, 1996). REGTOPP data usually consists of between 10 to 14 files (depending on the REGTOPP version), compressed into a zip file. These files include information about REGTOPP version, stops, lines, and time of validity. AtB (Sør-Trøndelag, Norway), Ruter (Eastern-Norway), Kolumbus (Rogaland, Norway) and Agder Kollektivtrafikk AS (AKT) (Agder, Norway) are examples of agencies using REGTOPP. With the advent of General Transit Feed Specification (GTFS) (Section 2.2.2), some Norwegian public transit agencies now provide both data sets side by side³.

2.2.2 The GTFS Standard

The General Transit Feed Specification (GTFS) is a format for public transit schedules developed by Google and Portland TriMet in 2005 (McHugh, 2013). It has now become the leading international standard for transmitting route data, primarily to Google Maps (Antrim et al., 2013).

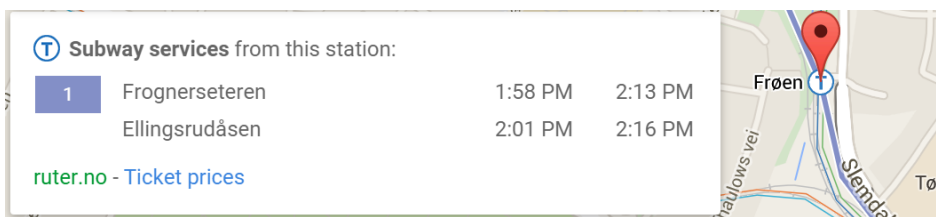


Figure 2.1: Google Transit in Oslo

³<http://next.kolumbus.no/2012/06/01/pne-rutedata-n-i-regtopp-1-2-format/>



The GTFS Data Exchange⁴ provides data from about 900 different agencies around the world, as of April 2015. In Norway, both Ruter and Kolumbus now provide GTFS data, which can be found on the GTFS Data Exchange and their individual home pages. Google Maps will display arrival times, stop locations, and provide users with directions that include public transit.

Finding transit information in Google Maps is easy, and can be done by clicking on a stop, with the results shown in Figure 2.1. In Oslo, and a few other cities, it will also highlight polylines following routes that pass the selected stop.

2.2.3 NeTEx

Network Timetable Exchange (NeTEx)⁵ is a CEN technical standard for exchanging public transit schedules, and can be seen as a compliment to SIRI (Section 2.3.1) (CEN/TC278/WG3/SG9, 2014). It aims for a high level of detail, with support for complex route patterns, holiday exceptions and ticket pricing. By using EXtensible Markup Language (XML), and the General Public License (GPL), they try to make the standard as approachable as possible. NeTEx supports all forms of public transit, including planes.

2.3 Real-time Vehicle Tracking Formats

Numerous public transit agencies have implemented real-time tracking systems. Without a standard unifying these implementations, many different Application Program Interfaces (APIs) were developed. Many have tried making a universal standard, with varying success.

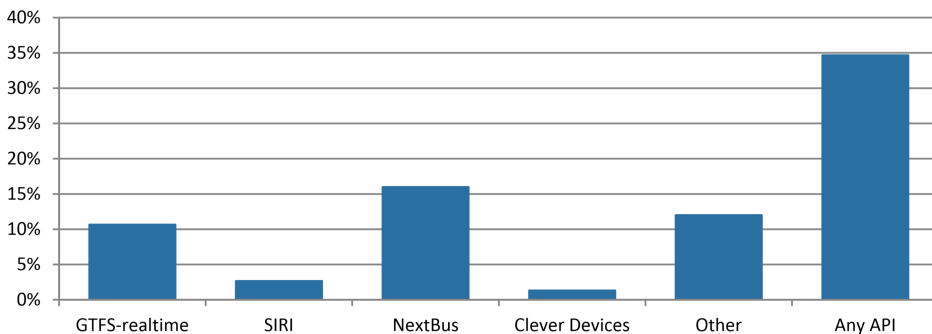


Figure 2.2: Results of APTA Survey: Arrival Time Formats (Grisby, 2013)

⁴<http://www.gtfs-data-exchange.com>

⁵<http://netex-cen.eu/>

In July 2012, the American Public Transportation Association (APTA) conducted a survey measuring the spread of real-time APIs in USA (Grisby, 2013). Of the 75 participants, only 16% lacked vehicle tracking capabilities, while the vast majority could track over 90% of their vehicles. However, only 37% of the agencies provide a real-time API or application. The survey also compared the different formats used for distributing real-time information. The results are shown in Figure 2.2. Keep in mind that these results are for American agencies, and may not reflect the world wide usage. SIRI is coming into wide spread use in Norway⁶, while NextBus (further explained in Section 2.3.2) is only present in Canada and USA⁷.

2.3.1 SIRI

Service Interface for Real Time Information (SIRI)⁸, is a CEN standard for communicating real-time public transit information between different devices (CEN/TC278/WG3/SG7, 2005). It was made in a collaborative effort by public transit agencies from many European countries, including France, the Scandinavian countries, United Kingdom and Germany. The standard is not owned by any one vendor, public transit agency nor operator.

To persuade public transport agencies to switch from their current systems, SIRI had to be exceptionally modern and usable. SIRI took on an open and modularized approach, to allow free customization and expandability. Furthermore, by using XML, SIRI can easily be utilized by anyone familiar with web development.

SIRI is made up of eight services that can be implemented according to the transport agency's needs and wishes. These services can be developed and updated independently of each other because of the modularity of SIRI. Central services of SIRI are listed below (Knowles, 2008).

Production Timetable Service (PT) returns the predicted timetables for a specified day, which can be used in an Automatic Vehicle Location (AVL) system. These timetables can easily be filtered by operator, line, and direction to only display the desired data.

Estimated Timetable Service (ET) provides real-time details about deviations from timetables. It also includes information about detours and cancellations.

⁶<http://bent.flyen.no/realtimeNorway>

⁷<http://webservices.nextbus.com/service/publicXMLFeed?command=vehicleLocations&a=lametro&r=2>

⁸<http://siri.org.uk>



Stop Timetable Service (ST) provides arrival times for stops, using timetables. It serves as a backup solution on information screens, when real-time is unavailable.

Stop Monitoring Service (SM) updates information screens and web APIs with real-time arrival estimates.

Vehicle Monitoring Service (VM) sends information about the location of vehicles, together with general line information. It can be used to display all public transit vehicles on a map, and gather detailed statistical performance data.

Connection Timetable Service (CT) provides schedule data for connected lines at a given public transportation hub. It is usually used combined with the CM service.

Connection Monitoring Service (CM) uses real-time arrivals, and schedules, provided by SIRI-CT, to coordinate interchanges. If one vehicle is delayed, connected vehicle routes are encouraged to wait to allow passengers to transfer.

General Message Service (GM) allows sending of arbitrary data like news related to the operation of the vehicles.

The services primarily used in this project are SIRI Stop Monitoring Service and SIRI Vehicle Monitoring Service, which are detailed below.

SIRI-SM

The SIRI-SM is usually the main motivation behind implementing a SIRI solution. It provides arrival time estimates for incoming public transit vehicles to a stop through an online API. These estimates are usually available on information screens at stops.

An excerpt from Ruter's SIRI-SM system can be seen in Listing 1. Ruter provides both XML and JavaScript Object Notation (JSON) versions of SIRI-SM, and the JSON version is included here as it is more readable. The arrival estimates are contained within *MonitoredCall*, and in this example the bus is 6 minutes and 6 seconds behind schedule. The listing presents one of the vehicles on line 74, and contains much more detailed information than NextBus (Listing 2) and GTFS-realtime (Listing 3).



```

{
  RecordedAtTime: "2015-02-23T13:38:04.276+01:00",
  MonitoringRef: "3010722",
  MonitoredVehicleJourney: {
    LineRef: "74",
    DirectionRef: "2",
    FramedVehicleJourneyRef: {
      DataFrameRef: "2015-02-23",
      DatedVehicleJourneyRef: "4307"
    },
    PublishedLineName: "74",
    DirectionName: "2",
    OperatorRef: "Norgesbuss",
    OriginName: "Mortensrud T [buss]",
    OriginRef: "3010951",
    DestinationRef: 3010047,
    DestinationName: "Vika",
    OriginAimedDepartureTime: "0001-01-01T00:00:00",
    DestinationAimedArrivalTime: "0001-01-01T00:00:00",
    Monitored: true,
    InCongestion: false,
    Delay: "PT366S",
    TrainBlockPart: null,
    BlockRef: "7007",
    VehicleRef: "330568",
    VehicleMode: 0,
    VehicleJourneyName: "50740",
    MonitoredCall: {
      VisitNumber: 20,
      VehicleAtStop: false,
      DestinationDisplay: "Vika",
      AimedArrivalTime: "2015-02-23T13:43:00+01:00",
      ExpectedArrivalTime: "2015-02-23T13:49:06+01:00",
      AimedDepartureTime: "2015-02-23T13:43:00+01:00",
      ExpectedDepartureTime: "2015-02-23T13:49:06+01:00",
      DeparturePlatformName: "2"
    },
    VehicleFeatureRef: null
  }
}

```

Listing 1: A Stop Visit Returned from Ruter's API

SIRI-VM

The SIRI Vehicle Monitoring Service (SIRI-VM) presents the positions of vehicles in traffic, and general information about them. It is ideal for online real-time maps, and on board displays. However, SIRI-VM is not as widespread as SIRI-



SM, and is only operational in a handful of cities. These cities include Tampere, Finland⁹ and New York City, USA¹⁰.

An excerpt from a vehicle location request for the bus service in Tampere can be seen in Appendix B. It is extracted from a list of all vehicles in transit in Tampere. An example of SIRI-VM usage in Tampere can be seen in Section 2.6.6.

2.3.2 NextBus

NextBus is an American standard for estimating and transmitting real-time public transit data (Schmier and Freda, 2002). As shown in Figure 2.2, it is a very popular standard in USA.

```
<vehicle id="7308" routeTag="2" dirTag="2_651_1" lat="34.051708"
  lon="-118.504486" secsSinceReport="50" predictable="true"
  heading="225" speedKmHr="53"
/>
<lastTime time="1431525204638"/>
```

Listing 2: NextBus Vehicle Location on Line 2 of the Los Angeles Metro

It is based on open URL access to information on stop data, line data, real-time predictions and vehicle locations. Through `nextbus.com`, it is possible to access this data from over 60 different agencies around USA and Canada, through a simple interface. A more developer-friendly interface is described in the public NextBus Public XML Feed documentation (NextBus Incorporated, 2013). It is a rather simple and straight forward format, as shown by the Los Angeles metro¹¹ example in Listing 2.

Even if NextBus has become widespread in USA and Canada, `nextbus.com` lists no European agencies. Because the author's are located in Norway, a European standard would be more suitable for this project.

2.3.3 GTFS-Realtime

GTFS-realtime¹² is Google's real-time expansion of GTFS (Section 2.2.2). It supports schedule updates, vehicle positions, and service alerts. The data is usually published using Google's Protocol Buffers¹³, which has the same features

⁹<http://lissu.tampere.fi/?lang=en>

¹⁰<http://bustime.mta.info/>

¹¹<http://webservices.nextbus.com/service/publicXMLFeed?command=vehicleLocations&a=lametro&r=2>

¹²<https://developers.google.com/transit/gtfs-realtime/>

¹³<https://developers.google.com/protocol-buffers/docs/overview>



```

{
  trip_id: "CR-Providence-CR-Weekday-Providence-Dec13-801",
  trip_name: "801 (6:20 am from South Station)",
  trip_headsign: "Providence",
  vehicle: {
    vehicle_id: "1806",
    vehicle_lat: "41.89791",
    vehicle_lon: "-71.35422",
    vehicle_bearing: "250",
    vehicle_timestamp: "1430824356"
  }
},

```

Listing 3: Excerpt of MBTA GTFS-realtime response

as XML, while being smaller and simpler. Listing 3 shows GTFS-realtime data for one of Massachusetts Bay Transportation Authority (MBTA)'s vehicles, in a JSON format.

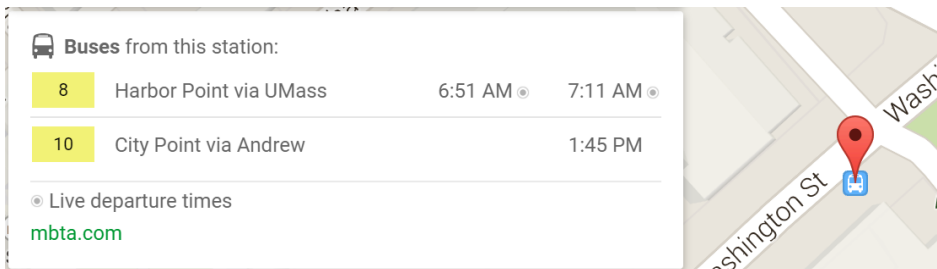


Figure 2.3: Google Maps Real-Time Arrivals in Boston

No Norwegian public transit agencies have started using GTFS-realtime, and instead rely on other standards like SIRI (Section 2.3.1). However, a few cities outside Norway have added real-time arrival estimates to Google Maps. An example can be seen in Figure 2.3, where the real-time arrivals are marked as *Live departure times*.

2.4 Google Maps

Google Maps is developed by Google. Because of Google's strong position in the online market, Google Maps quickly became the de facto online map service, and



now Google claim they serve over a billion users¹⁴. It exists on multiple platforms both as mobile applications and through their website. With continuous development, Google Maps has expanded to include functionality like street-view, real-time traffic, directions, and public transit.

2.4.1 The Google Maps API

At Google Maps' release, many third-party developers saw the opportunity to make the map into something more than just a map with driving directions. However, Google had not yet opened for such extensions, and the developers outside Google had to "hack" features into the map (Gibson and Erle, 2006). Seeing some of the results, Google decided to embrace third-party developers by opening an API for interacting with Google Maps.

The Google Maps API was released in June 2005, just four months after the initial release. Since then they have added features like, drawing polylines and polygons, custom map markers, and animated markers. Google has implemented slightly different versions for Android, iOS and Web.

2.4.2 The Google Directions API

The Google Directions API¹⁵, is Google's open directions distribution service. The API is used together with the Google Maps API, making it possible to input any location or coordinate, and get driving directions to any other location or coordinate. It will split the trip into various legs, each with an encoded polyline that follows the roads in Google Maps. Included in the response is a detailed description of every legs' distance, duration, and HTML encoded driving instructions. An example of a leg, as presented by the Google Directions API, can be seen in Listing 4. With the HTML instructions the Google Directions API can potentially be used to create navigation applications. For TransitVision, the primary values used are the polyline points, which is an encoded set of changes in latitude and longitude for each point along the leg¹⁶.

One of the limitations of the API, is that it is made for cars and will therefore ignore bus-only roads, which will sometimes result in ridiculous detours. Additionally, navigating to and from a coordinate underneath a bridge will produce unpredictable results, as there is no way of specifying a vertical coordinate.

¹⁴<https://www.youtube.com/watch?v=7V-fIGMDsmE&t=6m46s>

¹⁵<https://developers.google.com/maps/documentation/directions/>

¹⁶<https://developers.google.com/maps/documentation/utilities/polylinealgorithm>



```

{
  "distance" : {
    "text" : "84 m",
    "value" : 84
  },
  "duration" : {
    "text" : "1 min",
    "value" : 36
  },
  "end_location" : {
    "lat" : 59.9293528,
    "lng" : 10.7165649
  },
  "html_instructions" : "Head <b>southeast</b> on <b>Valkyriegata</b>
toward <b>Kirkeveien/Rv161</b>",
  "polyline" : {
    "points" : "caxlJs{k'APY^s@FMz@_B"
  },
  "start_location" : {
    "lat" : 59.9299391,
    "lng" : 10.7156175
  },
  "travel_mode" : "DRIVING"
},

```

Listing 4: One of the Legs on a Route Through Oslo

2.5 Server Technologies

A simple and lightweight web server was needed for the development of an API, which can be read about in Section 4.3. Numerous technologies for servers and databases were studied, focusing on the needs for TransitVision.

2.5.1 Node.js

Node.js¹⁷ is an open source, JavaScript runtime environment, used mostly for real-time web applications. Combining an event-driven architecture, the Google V8 JavaScript engine with the speed of C and C++, they achieved both low memory usage and performance (Tilkov and Vinoski, 2010). It was originally developed by Ryan Dahl, together with other developers working at Joyent, in 2009. However, its development is now overseen by the Node Foundation. Node.js' scalability and efficiency has made it popular among large companies like Microsoft, Walmart and PayPal. Meanwhile, its simplicity and heavy use of JavaScript

¹⁷<https://nodejs.org/>



make it very approachable to web developers and programming novices.

2.5.2 MongoDB

MongoDB¹⁸ is a schemaless, document-oriented, open source database, that is the most popular alternative to the well established relational databases, as of April 2015¹⁹. It can quickly store and retrieve JSON-objects, which is one of the main advantages when paired with JavaScript. MongoDB stores these JSON-objects in Binary JSON (BSON) files.

Being schemaless, one can easily add new collections in the database without setting up the columns beforehand. Additionally, it enables sudden changes to the columns of data stored, without causing issues.

2.6 A Brief Look at Similar Applications

A number of similar applications were examined to find their strengths and weaknesses. It was especially important to assess the quality of the various map applications, and how they worked. Additionally, the search for similar applications would also determine how unique TransitVision would become. Simply replicating an existing system is not as impressive as developing a distinctive approach.

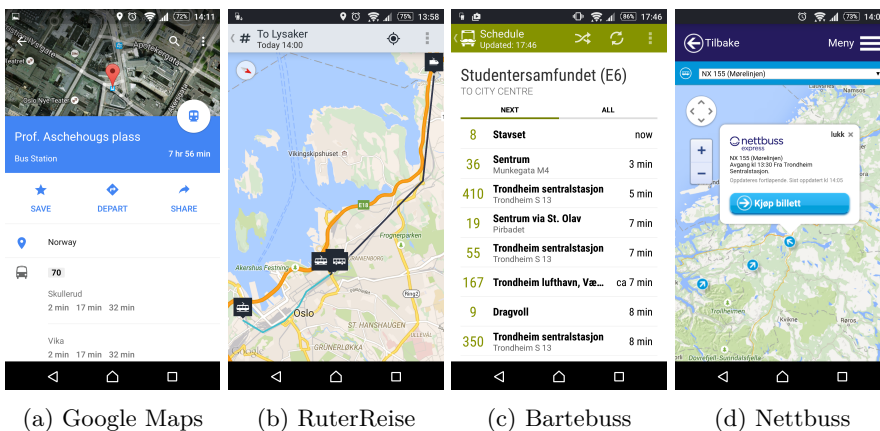


Figure 2.4: Public Transit Applications

¹⁸<https://www.mongodb.org/>

¹⁹<http://db-engines.com/en/ranking>



2.6.1 Google Maps

As discussed in Section 2.4, Google Maps is one of the largest transmitters of transit data. By incorporating GTFS data from hundreds of agencies, Google created a universal public transit schedule look-up. They support small-scale real-time look-ups, but have yet to grasp the potential of combining their maps with real-time position data. Some cities support drawing lines between stops, and displaying arrivals in real-time, but that is the peak of their abilities. Their main competitive edge is their large user base, and presence on all modern platforms. An example of transit data being displayed in the Google Maps Android application, is shown in Figure 2.4a

2.6.2 RuterReise

Ruter developed a simple and usable app, that allows real-time arrival look-ups for all of Ruter's stops. It has one of the simplest and intuitive user interfaces of the real-time departure applications. In addition to displaying the arrival times of lines at a given stop, it also has a travel planner. This planner can display your trip on a map, with straight lines from one stop to another, and is shown in Figure 2.4b.

2.6.3 Bartebuss

Bartebuss is a mobile application displaying public transit data in Trondheim. The main feature of the application is the real-time arrival estimates, which are displayed stop wise. This means the user searches for the desired stop, and then the arrival times at the stop are listed. Additionally, it includes the functionality of the Bus Oracle. An example is displayed in Figure 2.4c. The application can also be reached from the Web²⁰.

2.6.4 Nettbuss

Nettbuss is the largest bus agency in Norway, and served 137 million passengers in 2013 (Nettbuss, 2014). They have made a real-time map displaying their intercity buses, that is released both on web²¹ and as a smartphone application²². The Nettbuss Android application can be seen in Figure 2.4d.

²⁰<http://bartebuss.no/favoritter>

²¹<http://www.nettbuss.no/sanntid>

²²<https://play.google.com/store/apps/details?id=se.optidev.nettbussno>



2.6.5 Busskartet.no

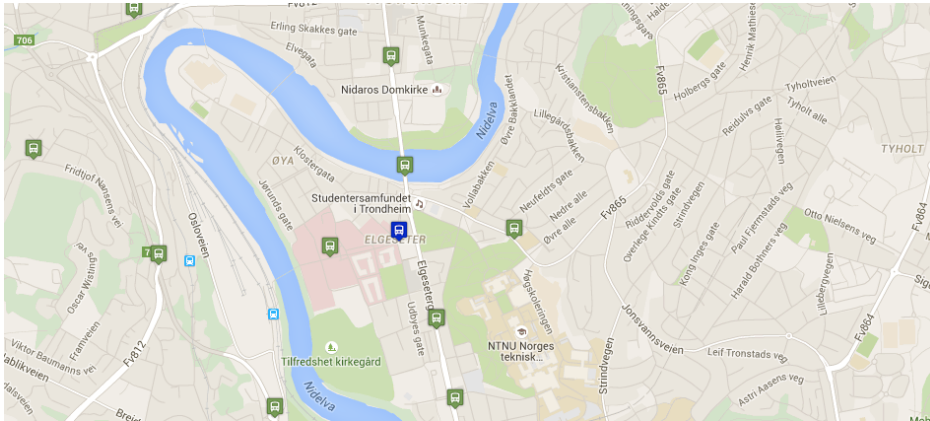


Figure 2.5: Busskartet Displaying Buses around Campus Gløshaugen

There are many different real-time and timetable based maps of public transit vehicles. One example of the latter is Busskartet²³ (The Bus Map), displayed in Figure 2.5, which currently is the only map covering AtB’s lines. It does not utilize AtB’s real-time system, but instead predicts vehicle movements based on the timetables and the Google Directions API. Its accuracy is thereby negligible, and Busskartet melts down to a curiosity, rather than a functional map.

The map’s primary achievement is the incorporation of the Google Directions API, which allows their vehicle markers to follow the correct roads. It is the result of a large amount of fine-tuning of the routes received from the Google Directions API. More about these issues can be read in Section 5.3.3.

2.6.6 Tampere Bus Map

A great example of real-time vehicle tracking is deployed in Tampere, Finland. Here they have successfully deployed SIRI-VM (Section 2.3.1), and created a map²⁴ displaying all buses and bus stops in and around Tampere. An image of the map is displayed in Figure 2.6. The dots on the map indicate bus stops, while the numbered circles are buses. It does not animate the buses, but instead updates their positions every three to five seconds, which results in stuttering movements.

²³www.busskartet.no

²⁴<http://lissu.tampere.fi/?lang=en>

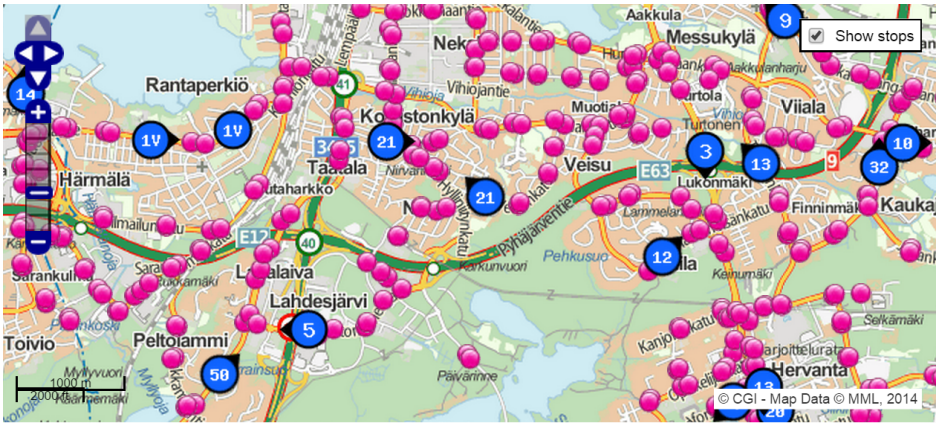


Figure 2.6: Tampere Public Transport's Traffic Monitor

2.7 The System Usability Scale

For a system to be successful, it must solve problems users have in a simple and understandable way. By using the System Usability Scale (SUS)²⁵, the usability can be measured using a predefined ten-item questionnaire. The participants will respond to each statement on a scale ranging from 1, strongly disagree, to 5, strongly agree. If a test person is unable to respond to a statement, that person should mark the middle of the scale instead of skipping it.

SUS was created by John Brooke (1996), and is constructed as a Likert scale. A Likert scale uses a technique for selecting items to include (in this case the statements) examples, which lead to extreme expressions of attitude (Brooke, 1996). Originally SUS had 50 questions, but through testing it was narrowed down to just 10. The final statements are listed in Appendix A.1.

2.7.1 Calculating the SUS Score

The way the SUS statements are worded, every even numbered question measures negative impressions, while the odd numbered questions are positive. Both of them impact the score with an equal amount, but there is a slight difference in how the score is calculated. For positive statements, the score contribution is the scale position minus one. For negative statements, the score contribution is five minus the scale position. This is explained clearer in Equation 2.1. These scores are then added together, and divided by the number of contributions to

²⁵<http://www.measuringu.com/sus.php>



find the mean value for each statement, as shown in Equation 2.2. When the mean response value is found for every statement, they are added together, and multiplied by 2.5 to make the result range from 0 to 100, as shown in Equation 2.3. X_n is the mean scale value for statement n , c is the total number of contributions and X_{ni} is the response i to statement n . Together, these three equations can be used to calculate the resulting score form a System Usability Test using SUS.

$$f(X_{ni}) = \begin{cases} 5 - X_{ni} & \text{if } n \text{ is even} \\ X_{ni} - 1 & \text{if } n \text{ is odd} \end{cases} \quad (2.1)$$

$$\text{Given: } f(X_{ni}), c \quad X_n = \frac{1}{c} \sum_{i=1}^c f(X_{ni}) \quad (2.2)$$

$$\text{Given: } X_n \quad SUS = \frac{5}{2} \sum_{n=1}^{10} X_n \quad (2.3)$$

Given the responses 1, 5, 2, 4, 5, 4, 4, 5, 4, 5 and 2 on statement one, which is 11 responses, it is possible to use Equation 2.2 combined with Equation 2.1 to calculate the mean answer. Since this is statement 1, $n = 1$. $f(X_{ni})$ is therefore $X_{ni} - 1$, which means Equation 2.2 becomes:

$$X_n = \frac{1}{11} \sum_{i=1}^{11} (X_{ni} - 1) = \frac{30}{11} \approx 2.73 \quad (2.4)$$

Equation 2.3 can be applied, after calculating the mean score of every statement in the survey. The final score computed by Equation 2.3 is the final score of the scale. Bangor et al. (2008) stated that a passable product should have above 70 points. Better systems score in the high 70 to upper 80, while products with a score above 90 are truly astounding. On the other hand, if a product gets a SUS score below 70, it is in dire need of improvements.





Chapter 3

Research and Development Method

During the autumn project, a preliminary study was conducted. The method used for this study will be described first in this chapter. Following, The chapter describes the methods used for TransitVision development and testing.

3.1 Preliminary Survey

Before developing TransitVision, the authors mapped travelers public transit habits through a web-based survey. The survey was distributed online using Facebook, and could thereby reach people with different ages and professions, from all over Norway. The survey was active from September 22nd until October 6th 2014, which gave the participants 14 days to answer the survey.

Questions were chosen based on research question 3, and information that could contribute in the development of TransitVision. These questions can be found in Appendix E.1, and includes questions about waiting time, frustration and schedule retrieval. Those using mobile applications as their main schedule retrieval method were given additional questions to map opinions on existing mobile applications.

The questions produce both quantitative and qualitative feedback, referred to as “Triangulation” by Jick (1979). Triangulation can lead to greater accuracy of the data gathered. In the preliminary survey, only the last question can be seen as qualitative. The participants were asked to write what they thought of the mobile application they used in a text box.

The survey maps travelers perceived waiting time. As travelers usually in-



correctly estimate their average waiting time with about two minutes (Watkins et al., 2011), the results may not reflect their actual waiting time. However, reducing travelers' perceived waiting time could have greater impact on traveler satisfaction, than their actual waiting time.

By spreading the survey on the Web, there is no way to guarantee that all traveler groups are represented. One of the problems with web surveys is the exclusion of travelers without Internet connection (Couper, 2000), and is therefore unable to create a complete picture of traveler habits. Additionally, as Porter and Whitcomb (2003) describe it, people using the Internet are more interested in technology than the general public, and may skew the results. A participant may submit multiple responses to the survey, as there is no protection against it. Requiring an email address could help minimize the issue, but it could also negatively impact the number of participants as some are uncomfortable with sharing their email address.

3.2 Finding and Prioritizing Requirements

Brainstorming (Diehl and Stroebe, 1987) was used for creating requirements. It is a method where the team members write down all ideas that come to mind, and revise them afterwards to define the final requirements. By using brainstorming, many ideas were generated and could be evaluated as final requirements.

Initially, the authors utilized the Scrum methodology to structure the work. A product backlog was created and hosted on Jira¹, and it kept track of all user stories and tasks. Scrum (Sims and Johnson, 2014) allows developers to always know what the rest of the team is doing, and what they should do next. However, as the authors grew more confident with the requirements of the application, the scrum methodology became redundant and its use slowly faded. Moreover, the project had no customer. The product backlog still formed the base for what functionality to implement, and their priority.

Basic ideas and requirements for the application had to be set to make sure the application was suited for the problem description. It was important to make the application user friendly, while representing the real-time data in an adequate way. The requirements were created as a list containing the requirement ID, a description, and a prioritization. The most important requirements were implemented first, to ensure the completion of the most essential functions.

¹<https://www.atlassian.com/software/jira>



3.3 Designing TransitVision

A sequence diagram, a class diagram, a flow chart and prototypes form the design of TransitVision. Every element helped unifying the thoughts of the authors, which made planning the development much easier.

A sequence diagram describe the communication between entities in the system (Li et al., 2004), a class diagram describe the code structure of the system and the relationship between objects², and the flow chart described the relationship between screens in the user interface³. Together they formed the structure of the code, and defined what was needed in the code to realize the requirements. Additionally, the diagrams form a description of the systems architecture, which makes it easier for outsiders to understand the structure.

The user interface was prototyped to plan the user interface design. Element sizes, shapes and colors can be drawn on paper or a computer, making it easier for a designer or developer to see which designs work best in practice. A prototype can be used to test and refine the design more efficiently than using a coded prototype (Snyder, 2003).

An example of an important aspect to be aware of when creating a drawn prototype is button placement and size. Buttons having the same function should look the same. For example, all delete buttons should be colored red, and read the text “Delete” wherever they are located within the application. The Android Design Principles⁴ were strictly followed when developing TransitVision. These design principles are created to make all Android applications work the same way, and by that seem familiar for users.

By creating different prototypes and testing the design, it is easier to make sure the real-time data is presented in a satisfactory manner. Both digital and paper prototypes were created for TransitVision to ensure the design, but they were never tested on real users. Testing the prototypes on users would have made the designs evaluated by a third party, which might have discovered flaws the authors did not.

3.4 Testing Server Compatibility

Position estimation requires asking for arrivals on every stop on a line. This may produce short bursts of requests to the servers of transportation agencies, whenever a position query is executed. Some of these agencies, have implemented barriers to prevent Distributed Denial of Service (DDOS) attacks (Lau et al., 2000), and this barrier presents one of the primary interoperability issues. In

²<http://www.agilemodeling.com/artifacts/classDiagram.htm>

³<http://www.breezetreel.com/articles/what-is-a-flow-chart.htm>

⁴<https://developer.android.com/design/get-started/principles.html>

some cases, it could take several seconds for TransitVision to gather enough real-time data to draw vehicles.

To provide a satisfactory user experience, the limitations of the various SIRI-SM systems had to be determined. Additional measurements were taken to pinpoint whether two machines communicating with the SIRI system would interfere with each other. The test itself was conducted by sending as many requests as possible from two different locations, for one minute. First individually, and then simultaneously. Both small and large packages were used to measure whether the system was limited by number of requests, or by the generated traffic.

The machines used were placed at two different locations, one in Trondheim, and one in Amsterdam. Both machines had a 100 Mb/s Internet connection at their disposal. Neither their Internet speed, nor their hardware should produce a mentionable impact on their performance in this test, as the amount of traffic sent is so low. Only Kolumbus' and Ruter's systems were tested this way, as they were the primary focus of the interoperability study.

SIRI only defines which data should be included, and does not specify the format of the data. Consequently, different transportation agencies use different coordinate systems, and time formats. To further measure the interoperability of TransitVision, many SIRI implementations were examined, by comparing the values in the data packages received from these SIRI implementations. Six systems were examined in total.

3.5 User Test

Result validation was done through a SUS survey, combined with users testing the application. The result from the SUS gives a quantitative measurement of the TransitVision usability. This is required to interpret whether or not the application present the real-time data well.

The test the participants have to complete before answering the SUS questionnaire was a scenario test (Kaner, 2003). Participants were presented with the the application and the task to find information on their favorite line. If they had no favorite line, they got a suggestion.

To ensure the reliability of the SUS score, there should not be too few people included, while not becoming too time consuming because of a large number of participants. Faulkner (2003) discusses the need to test usability on more than five participants. It was assumed that five participants sufficed, but Faulkner disproves that by showing many problems within an application were not recognized by only five people. Only 85% of the problems were discovered on average. By doubling the size of the group to 10, the average problems discovered became 95%, which is a significant difference. A set of 15 users found even more (about 97.5%).



Tullis and Stetson (2004) take a different approach, with 123 participants testing two different websites. They discovered that one of the websites was significantly preferred over the other. The paper shows how few participants were needed before this trend was revealed, by comparing the results of subsets ranging from 6 to 14 testers, of the total 123 participants. With a sample size of 12 participants, the System Usability Scale yielded a 100% “correct” conclusions, meaning that only 12 participants were needed to indicate that one website was significantly better than the other. Because of these two papers, the user test of TransitVision would include at least 12 subjects.

A range of different types of users, different ages, and professions represented by the testers was important. The age ranges from 20 to 60, and the professions range from students, to health personnel to engineers. By this, the responses represent different types of users within the target group.

To maintain the meaning of the statements, the questionnaire was conducted in English, instead of translating them to Norwegian. Finstad (2006) discovered that 9 out of 18 (or 50%) non-native English speakers had a hard time understanding the word “*cumbersome*” in statement 8. However, because one author would always be present when participants answered the questionnaire, difficult words could be explained if necessary. When asked, “*cumbersome*” would be translated to the Norwegian word “*tungvint*”.



Chapter 4

Results

A preliminary study into travelers' public transit habits helped form the result of TransitVision. The server back-end function as an adapter between different SIRI implementations and the mobile application. After implementation, the application was evaluated by a group of testers through SUS.

4.1 Results from the Preliminary Survey

The preliminary survey had the purpose of identifying traveling habits of public transit passengers. The results were expected to reveal the passengers using mobile applications as the travelers with the shortest waiting time. However, the data gathered contradicts this thought.

4.1.1 Waiting Time

The majority of the survey participants, 81%, thought that having the opportunity to see the bus' location would have a positive impact on their waiting time. 11% resisted the idea, for unknown reasons. Nevertheless, the results confirmed that the passengers desire a transit map application. If the participants had shown more resistance to the idea, it would have to be reworked.

Of the people traveling by bus, 65% thought they waited more than five minutes for the bus to arrive (numbers in Figure 4.1). It is worth mentioning that the waiting time data gathered is the participants own perceived waiting time, and may not reflect the actual time spent waiting (as discussed in Section 3.1).

One of the questions posed in the survey dealt with reasons for missing the bus. As shown in Table 4.1, the responses are evenly distributed, but one alternative stands out; people have a hard time calculating how long it takes to get to the



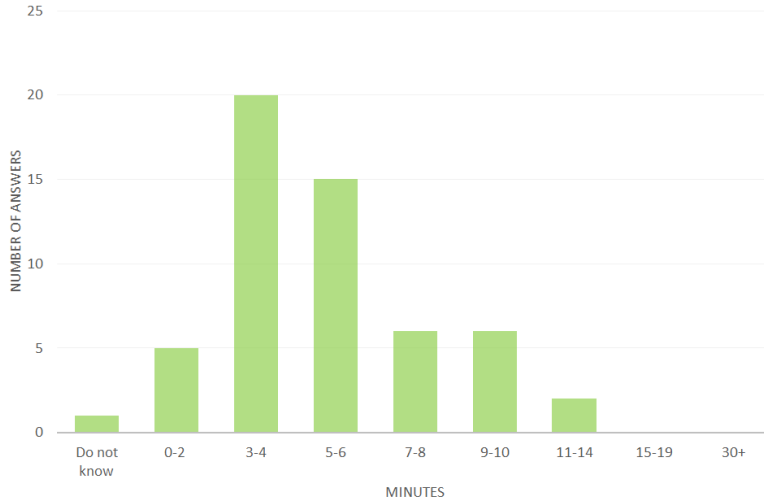


Figure 4.1: Perceived Waiting Time

bus stop. It is possible to create some kind of warning system that will alert the passenger when it is time to leave.

As depicted in Table 4.2, the average perceived waiting time for travelers using modern mobile applications, is about the same as the travellers using other means of schedule retrieval. Furthermore, the users of web applications believe they wait an average of 0.7 minutes (≈ 42 seconds) longer than mobile application users. This difference in waiting time might stem from the fact that web users are the least active bus users, with only 2.1 trips per week on average. Consequently,

Table 4.1: Reasons for Being Too Late for the Bus

Reason	Count
I am never too late	3
The bus was ahead of schedule	4
Got incorrect route information from my app	5
Real-time was inaccurate and showed the wrong time	10
Miscalculation of walking time to the bus stop	20
Did not use route information	6
The bus departs so frequently that I do not care if I am late	6
Other	1



Table 4.2: Average Percieved Waiting Time by Route Retrieval Method

Retrieval method	Count	Waiting time	Trips/week
Web application	17	5.7 min	2.1
Mobile Application	27	5.0 min	4.3
Other	11	4.9 min	6.7
Total	55	5.2 min	4.1

they might arrive too early at the bus stop because of their inexperience, and as a result, they wait longer. Additionally, those that travel less by bus might not want to go through the inconvenience of installing an application, or finding a timetable, for just one trip. Either way the frustration level is about the same for all sorts of travelers, as shown in Table 4.3.

Table 4.3: Average Frustration by Route Retrieval Method

Retrieval method	Mean frustration level
All	5.8
Web application	5.5
Mobile Application	6.0
Other	5.8

The frustration level produced by waiting for the bus was measured on a scale from 1 to 10, where 10 is extremely frustrated. Figure 4.2 shows the frustration level distribution as a bar chart. No one was completely content with their time spent at the bus stop, as none of the participants answered “1”. On the other hand, the figure shows as many as five participants reach their maximum level of frustration by simply waiting for their bus. These results indicate that there is a lot of room for improvement on traveller satisfaction.

4.1.2 Application Usage Feedback

The survey also included a section on web and phone application usage, to map their popularity. Of the 55 participants, 27 answered that they primarily used mobile applications when finding route information. As shown in Figure 4.3, the most popular applications were Bartebuss in Trondheim, and RuterReise in Oslo, which are two popular applications with many different features. By combining the best features from these two, TransitVision should have a fair chance to compete in the market. A more thorough description of RuterReise is found in Section 2.6.2, and of Bartebuss in Section 2.6.3.

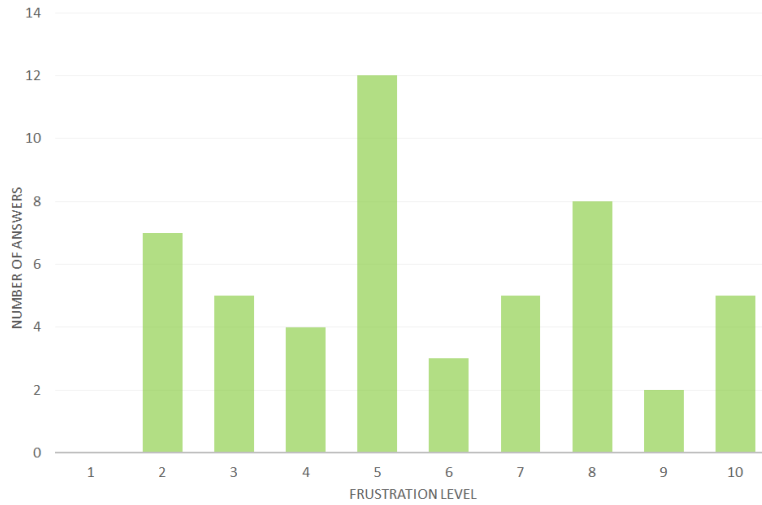


Figure 4.2: Frustration Level for Public Transit in Trondheim

Participants were encouraged to comment on their favorite features in the application they use, and which features it lacked. About RuterReise, people

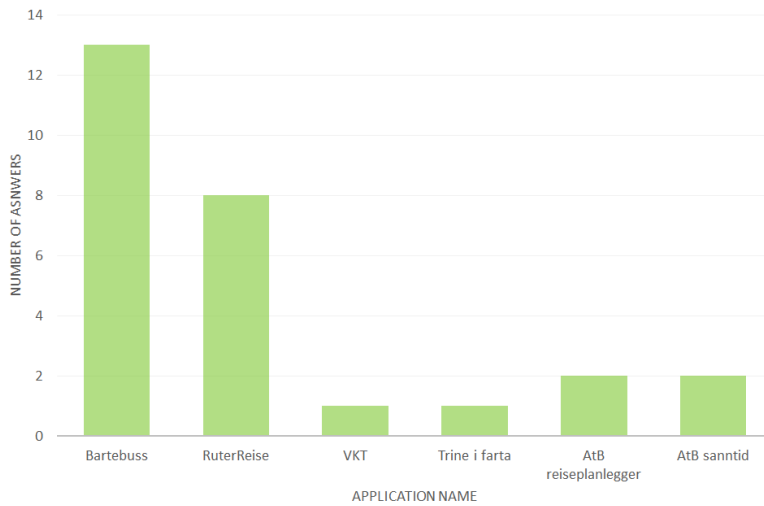


Figure 4.3: Popularity of Mobile Transit Applications



commented that the application needed functionality for showing the whole trip, and not just when the bus will pass the next stop. RuterReise does support displaying the whole route through the trip planner, but it might not be visible enough. Other than that, RuterReise's comments were mostly positive. According to the feedback on Bartebuss, the application tends to sporadically update arrival times, resulting in the bus arriving earlier than what the real-time suggested. This may be caused by inaccuracies in AtB's real-time system, and not because of Bartebuss. Another person commented that finding the timetable for a given bus route is too cumbersome, and that it should have a list of lines in addition to their list of stops.

4.2 TransitVision Requirements

A list of requirements were created to capture all features needed to realize TransitVision. The final requirements are listed in Table 4.4. The first eleven requirements are intended for the mobile application, while the last eleven are for the API.

In the beginning of development, the mobile application included three main screens: 1) a map screen, 2) a screen containing a list of stops, and 3) a BusTUC Oracle screen. Because the map screen is essential for answering the research questions, M1, M2, M3 and M11 are the most important requirements for the mobile application. M10 is also a highly prioritized requirement, because navigation is important for the usability of the system. Requirements regarding the list of stops and the Oracle are of secondary priority.

The API should be a simple interface for relaying position estimates, together with stop and line information. Its main functionality is represented by requirement S4, and most of the requirements are associated with it. S1 and S8 are related to information gathering, while S2, S3, S5, S6, S7, S10 and S11 deal with information distribution. S9 concerns the architecture, and requires setting up a remote server dedicated to TransitVision.

4.2.1 Quality Attributes

Many quality attributes were considered for TransitVision, but three were selected as the most important attributes. These attributes are listed below.

Interoperability Bass et al. (2012) defines interoperability as how well two systems can communicate, often through an interface. Research Question 1 (Section 1.2) asks how easily different SIRI implementations can be incorporated, and TransitVision's interoperability is therefore considered of great importance.



Table 4.4: TransitVision Requirements

ID	Description	Priority
M1	The mobile application should display a map	High
M2	The map should animate moving buses, using real-time data from the API	High
M3	The map should display all bus stops managed by a given company	High
M4	It should be possible for users to look up stops and see when the next bus arrives	Medium
M5	Users should be able to ask the BusTUC Oracle for schedules	Low
M6	The users should be able to create a list of their favorite bus/tram/subway lines	Medium
M7	The users should be able to create a list of their favorite stops	Medium
M8	The application should list stops by distance from the user	Medium
M9	When clicking on a vehicle on the arrival list, that vehicle's location should be displayed on the map	Medium
M10	The application should allow easy switching between the different fragments	High
M11	The user should be able to search for a desired line	High
S1	The Web-API should gather info about all stops and lines, and store it in a database	High
S2	The Web-API should present a list of all bus stops	High
S3	The Web-API should present which bus stops serve which lines	High
S4	The Web-API should present the locations of buses on a given line	High
S5	The Web-API should convert all coordinates to lat/long	High
S6	The Web-API should list all available bus operators	High
S7	The Web-API should present all data on JSON form	High
S8	The Web-API should be able to read standard SIRI-SM XML files (Ruter can convert to JSON automatically)	Medium
S9	The Web-API should be deployed on a remote server	High
S10	The Web-API should provide positions of vehicles in an area	Medium
S11	Position estimates should be placed on the road	Medium



Portability Bass et al. (2012) defines portability as how easily software built for one platform can be changed to run on a different platform. As TransitVision aspires to become a popular application, it should be available on as many platforms as possible. TransitVision should be portable, as long as it does not affect interoperability.

Usability According to Bass et al. (2012), usability is measured by how easily a user can accomplish a desired task by using the system. Usability was important for TransitVision to determine whether it satisfied users as a representation of real-time data, which is Research Question 2.

4.3 The Technologies Behind TransitVision

Technologies have to adhere to the requirements set in Section 4.2, while enabling rapid development because of the limited time frame. The authors primarily looked at familiar technologies, but also explored alternatives outside their area of expertise.

4.3.1 Targeting a Platform

By building a web application, it is possible to target a broad user base, as every platform can use it. Tools, such as Phonegap¹ or SenchaTouch², can be used to “wrap” web application into a “native” mobile applications, making it possible to develop for several mobile platforms simultaneously using HTML5, CSS and JavaScript. However, issues arise regarding performance and design when compared to native mobile applications (Charland and Leroux, 2011). The authors have experience with actual native applications being better suited for their respective platform than a Phonegap/SenchaTouch application.

Android OS had a worldwide market share of 81.5%³ in 2014. One of the reasons for this is because Android is available on numerous smartphones, in all price ranges (Mahapatra, 2013). iOS is developed by Apple and is exclusive to iPhones and iPads. Even though Mahapatra states that iOS is the most popular operating system in Norway at 56%, Android is not far behind at 41,41% at the end of 2013.

Because Android is Java-based, it is more approachable than iOS development, as the authors have more Java experience, but little to no experience with iOS’ Objective-C or Swift. According to Goadrich and Rogers (2011), the bar for Android development is usually a lot lower than iOS development.

¹<http://phonegap.com/>

²<http://www.sencha.com/products/touch/#overview>

³<http://www.idc.com/getdoc.jsp?containerId=prUS25450615>



One of the major differences is the cost of development for each platform. The development team has both Android and iOS phones available, so the hardware costs will not be discussed. However, while deploying an application on an Android device is free of charge and easy to do, Apple requires a developer licence. The Apple Developer Licence cost \$99/year⁴, and without it the application can only be run in a simulator.

Given the limited time frame of this project, it is perhaps unwise to choose too many unfamiliar technologies. Combined with development costs, and platform openness, it is natural to select Android as the primary platform.

4.3.2 Selecting a Server Framework

The selected server framework for the TransitVision API, had to be able to solve requirement S5 (coordinate conversion), S7 (send JSON-objects) and S8 (read XML). Most server frameworks should support these features, but having tools available out of the box could have a great impact on development time.

Node.js (Section 2.5.1) has all these features either included or in their vast package manager. As node.js is a JavaScript framework it can easily send JSON-objects, without the need for external libraries. The Coordinator package⁵ allows quick conversion from the Universal Transverse Mercator coordinate system (UTM) and many other formats, to latitude and longitude. Numerous XML-parsers are also available⁶.

Asp.net was one of the alternatives to node.js, as it can fulfill all the requirements. It is primarily made for windows computers, but can run on Linux computers by using Mono⁷, or by using a docker⁸. Because of all these underlying frameworks, the initial setup is tedious when compared to node.js.

Java is also one of the logical choices when it comes to web development, as the authors has previous experience using Java. However, Java needs external libraries to enable JSON interaction, as required by S7.

Node.js was eventually chosen because of its simplicity, performance and JSON integration.

4.3.3 Selection of Database Technology

The only requirement directly linked to data storage is S1 (see Figure 4.4), while S2, S3 and S7 are related to formatting and distribution of stored data. S7

⁴<https://developer.apple.com/programs/start/ios/>

⁵<https://www.npmjs.com/package/coordinator>

⁶<https://www.npmjs.com/search?q=xml>

⁷<http://www.mono-project.com/docs/web/aspnet/>

⁸<http://blogs.msdn.com/b/webdev/archive/2015/01/14/running-asp-net-5-applications-in-linux-containers-with-docker.aspx>



requires data to be in JSON-format, and it could therefore be useful to fetch JSON-objects directly from the database.

MongoDB goes hand in hand with node.js, and is the most popular alternative to the relational databases (see Section 2.5.2). By storing the files in the BSON format and exporting them as JSON objects, TransitVision can directly distribute the values from the database. CouchDB⁹ is very similar to MongoDB, as they are both document-oriented NoSQL databases using JSON-objects. It is possible to combine a relational database with JSON-objects, as shown with a proof-of-concept by (Chasseur et al., 2013). However, it was determined a combination of MongoDB and Node.js would be the best approach for TransitVision, as these are official tools which the authors have previous experience with.

4.3.4 Map APIs

When it comes to map solutions for Android, the most commonly used is the Google Maps API (Section 2.4.1). There are many alternatives to the Google Maps API on the Android platform, such as Here¹⁰, Mapbox¹¹ and MapQuest¹². However, the Google Maps API is the most established map solution on Android and was actively used by 54% of smartphone users in 2013¹³. Users should therefore already be familiar with the look and feel of Google Maps.

Google Maps API support all the requirements set for the map, such as drawing multiple custom markers. It also supports user interaction with markers, enabling detailed information about stops and vehicles to be displayed. Additionally, the authors have previous experience with the Google Maps API, but lack experience with the other map solutions.

4.4 System Architecture

To fulfill the requirements and quality attributes set in Section 4.2, TransitVision had to be built on a fitting architecture. This includes selecting an architectural pattern and code structure.

⁹<http://couchdb.apache.org/>

¹⁰<https://developer.here.com/>

¹¹<https://www.mapbox.com/>

¹²<http://developer.mapquest.com/web/products/featured/android-maps-api>

¹³<https://www.globalwebindex.net/blog/top-global-smartphone-apps>



4.4.1 Architectural Pattern

The initial idea was to have the mobile application do everything from information gathering, to position estimation. However, the system needs to know the travel time between every stop on a line to provide accurate position estimates (this will be further discussed in Section 4.6.2). When few vehicles are active on a given line, TransitVision will not always be able to find travel times between all the stops on the line. In those cases, it will rely on travel time data from earlier position queries. By storing these travel times on a server, every user would be able to utilize them.

By employing the Service-Oriented Architecture (SOA) pattern (Bass et al., 2012), TransitVision can extend its interoperability to include other SIRI implementations, or potentially different standards like GTFS-realtime. Figure 4.4 shows the simple SOA architecture used in TransitVision. This pattern splits TransitVision into a back-end API and a front-end Android application. This structure will only present users with the data they need to draw the vehicles. Additionally, it will allow caching of responses and send the same data to multiple users, thereby relieving traffic from the SIRI servers. A centralized API fosters portability, as individual platforms will simply have to render the position estimates received from the API.

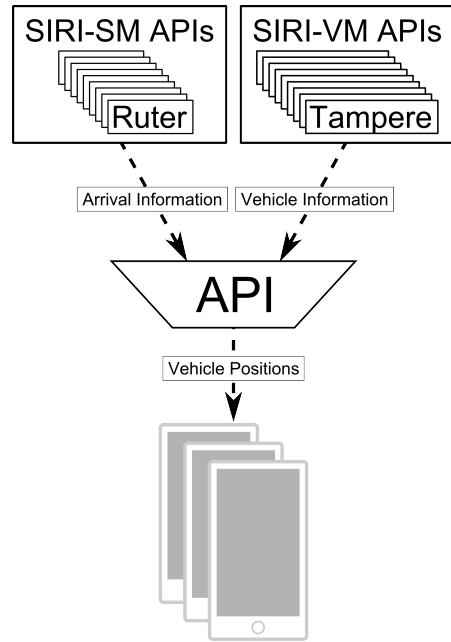


Figure 4.4: TransitVision Architecture

4.4.2 Class Structure of Mobile Application

A class diagram describes the code structure of the mobile application. The diagram shows all classes, their attributes and methods, and their relationship. The diagram can be used to easily understand the relations and code structure for current and future developers.

Figure 4.5 shows the class diagram for the mobile application. At the top of the figure is the activity, containing two fragments, and communicates with them through an interface. Most of the functionality is contained within the fragments,



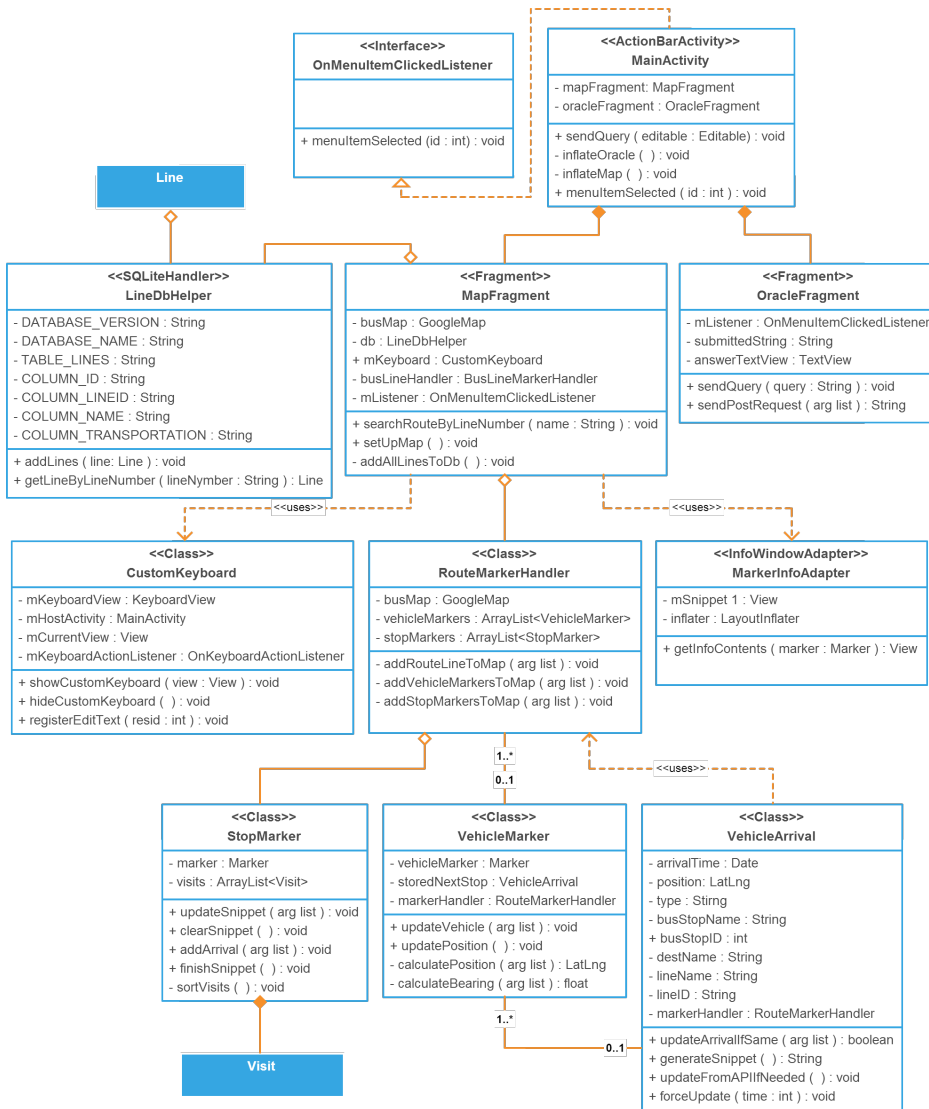


Figure 4.5: Final Class Diagram for the Mobile Application

which can be seen in the figure as well. The class *LineDbHelper* stores all lines in a local database on the phone, which makes it possible to check whether the line the user searches for exist or not.



4.5 Design

This section presents the design choices for TransitVision. This includes a sequence diagram, technique used for calculating vehicle positions, prototypes, icons, search tools, and a flow chart.

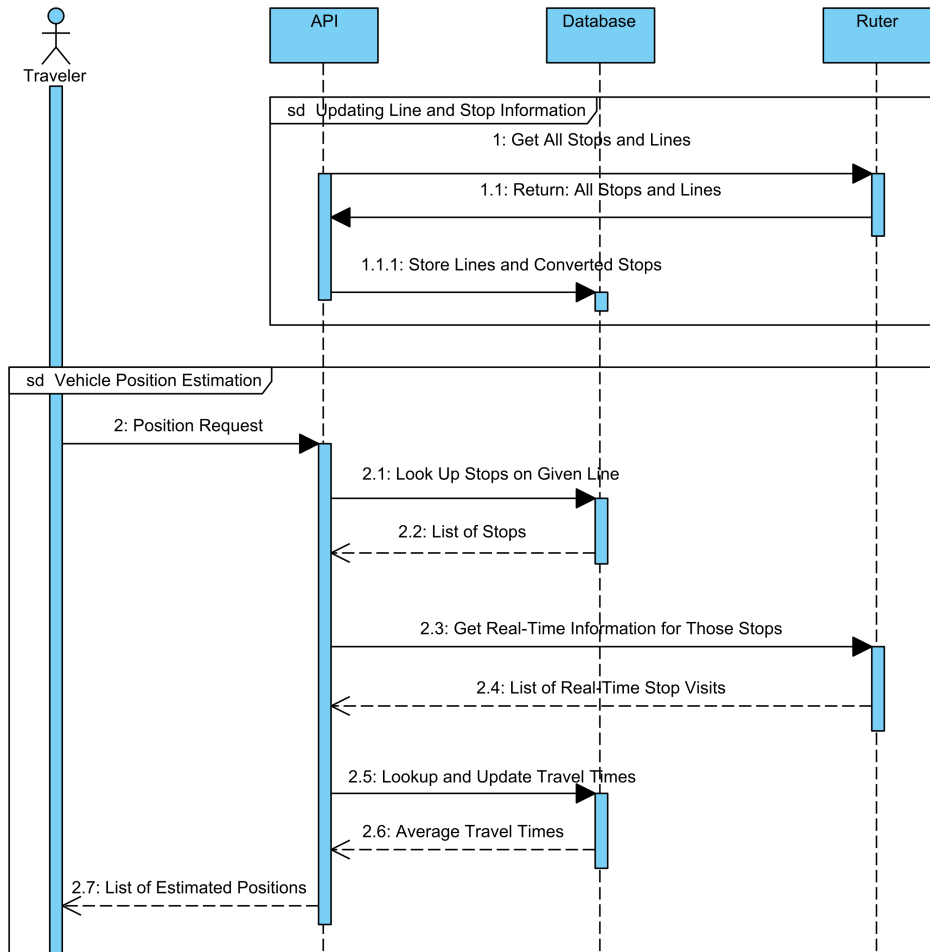


Figure 4.6: Interactions Between User, API, Database and Ruter



4.5.1 Estimating Positions

As discussed in Section 4.4, the primary function of the API is to present position estimates. This process consists of several steps, which require different data. These steps are listed below and detailed in Section 4.6.2.

1. Update stop and line information.
2. Wait for a position request.
3. Query all stops on the given line for real-time information.
4. Analyze all arrivals, extract the necessary information and store it in the database.
5. Calculate vehicle positions.
6. Return a list of all vehicles on the line, with their estimated positions.

To estimate the positions, the API uses a combination of stop positions and estimated arrival times. As shown in Figure 4.6, and the list above, the API will first update lines and stops before storing these in a database. It will then wait for requests from the front-end. One of which is the request for vehicle positions on a specified line. Once the API receives such a request, the server will find all stops on that line, and query the agency for real-time data on those stops. Using this data, and data from previous real-time queries, the API will attempt to estimate the expected travel times between all the stops on the line. By combining these estimated travel times, with the real-time arrival data, the system can create and return a list of estimated vehicle positions.

4.5.2 Mock-Ups

Prototype mock-ups are tools used when developing user interfaces. Mock-ups promote easy design testing, and are most commonly combined with user testing¹⁴. Prototypes for TransitVision were primarily used to decide which design elements to include, and not to test it on users. Chosen elements were type of menu, and colors to use in the implemented application.

Paper Prototype

The first prototype was created on paper using a pencil and an Android screen template¹⁵. It focused primarily on navigation within the application. Figure 4.7

¹⁴<http://www.paperprototyping.com/what.html>

¹⁵<http://www.androiduipatterns.com/2012/09/ui-sketching-on-paper-templates.html>



displays three different menus: a Navigation Drawer, a Tab Menu and a Spinner Menu. The spinner menu was quickly discarded because it is not commonly used, and may seem unfamiliar to users, and is more fitting when filtering data. For instance, when switching from a weekly view to a yearly view in a calendar¹⁶.

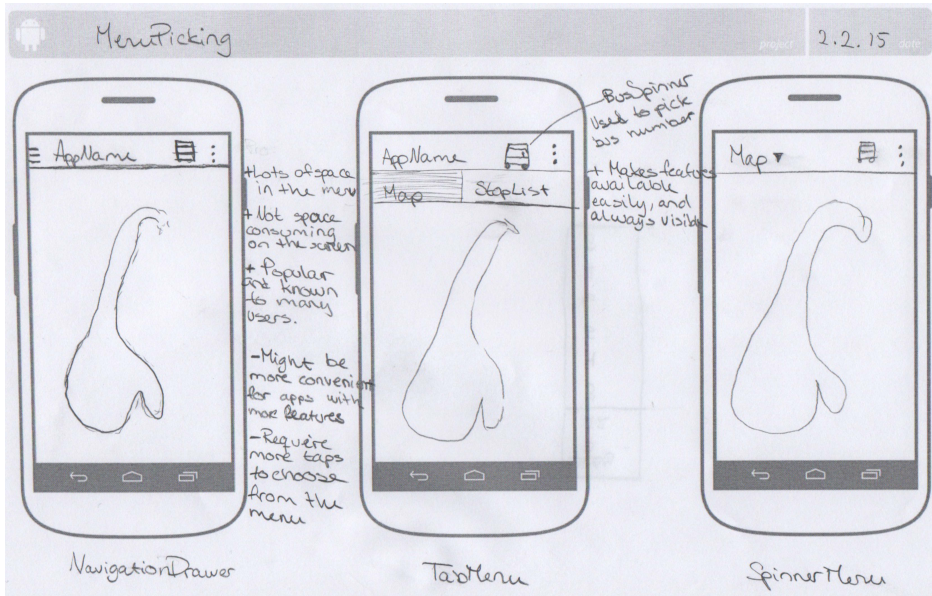


Figure 4.7: Navigation Drawer Menu, Tabbed Menu and Spinner Menu

More detailed drawings were created based on the decision to discard the Spinner Menu. The prototype deemed the tabbed menu unfit, due to an additional tabbed menu required for the list of stops; two tabbed menus on one screen is not good practice. The navigation drawer was chosen as a menu for the TransitVision.

Digital Prototype

The digital prototype contain the same elements as the paper prototypes, as displayed in Figure 4.8. However, the digital mock-ups take the design one step further by introducing colors, and more “real” screens and components. It is thus easier to settle on which components are needed to realize the user interface design, and which color combinations work well together. The presented mock-

¹⁶<http://developer.android.com/guide/topics/ui/actionbar.html#Dropdown>



ups are visually clean and contain few components simultaneously. Colors used are mainly black and green, in addition to grayscales.

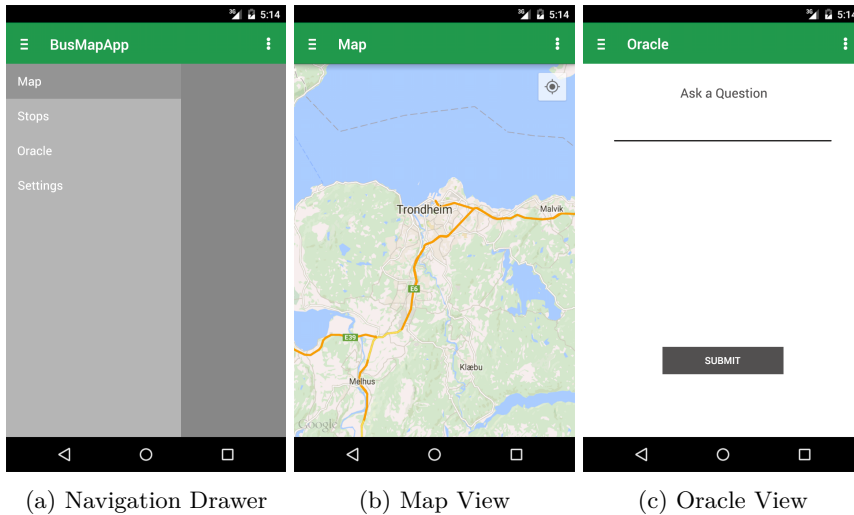


Figure 4.8: Mockup Screens

The color chosen for the action bar is green. Other than being aesthetically pleasing, its association with environmentally friendliness is related to using public transit, and thus is a fitting color.

4.5.3 Line Search Tools

To interact with the map, the user has to search for the desired line. This is done through the search field above the map. Tapping it will reveal a custom keyboard where the user can type the line number.

The search field was initially a collapsible `SearchView` menu item appearing in the action bar. It is illustrated in Figure 4.9 as the magnifying glass on the right side of the action bar. When tapped, the magnifying glass would slide to the left and reveal a search field. However, a regular text field (or `EditText`) replaced it because of incompatibility with the custom keyboard. The search field is visible in Figure 4.10b.



Figure 4.9: Collapsed `SearchView`

The Android soft keyboard is used for regular typing on the phone, such as text messages or Internet surfing. Other standard soft keyboards exist, like

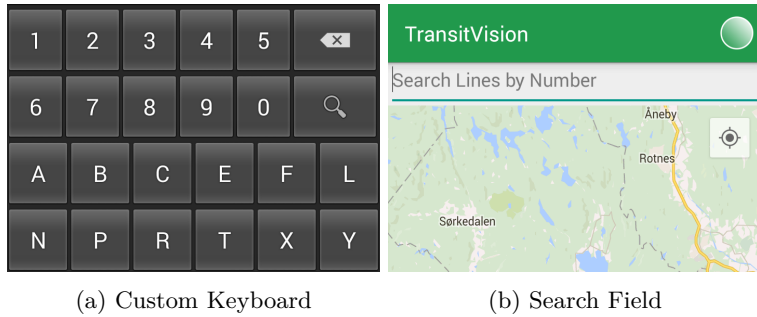


Figure 4.10: The Search Components

a keyboard that only uses numbers for dialing phone numbers. Figure 4.10a displays the keyboard used for Ruter. The keyboard consists of all digits, and a set of letters, selected based on the Ruter database of line numbers. The letters discovered in Ruter’s lines are A, B, C, E, F, L, N, P, R, T, X and Y.

4.5.4 Marker Icons

The marker icons representing stops and vehicles were difficult to create. It is important that the user understands what the icons mean as quickly as possible, by making them as intuitive as possible. This will also promote usability, given that the user understands the icons better. How intuitive the icons are depends on what kind of icons already exists, and which of these the user is familiar with.



Figure 4.11: Early Icons

An example of the early icons created are illustrated in Figure 4.11. The figure displays three of the icons used in the beginning of the development. These icons were supposed to show what kind of transportation type the line were, and where the stops were located. After showing the icons to others, they did not understand which icons were the stops, and which were vehicles. Another disadvantage with



these icons is they do not illustrate which direction the vehicle is moving. Because of this, new icons were created which clearly differs between what is a stop, and what is a vehicle, displayed in Figure 4.12. This icon collection does not show the difference between transportation types.

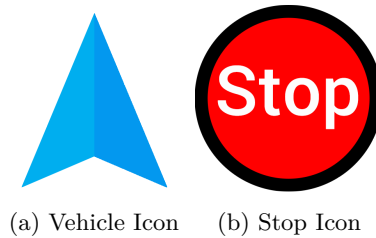


Figure 4.12: Final Icons

The vehicle marker is inspired by GPS navigation arrows, which makes it more associable with a moving vehicle than the older icons. The marker is animated to show the vehicle's position estimated with the real-time arrival data gathered from Ruter. Additionally, the marker rotates to indicate the vehicle's bearing.

The stop markers, on the other hand, are not moving. They are pinned on the map when the request is sent. The icon is inspired by a traffic stop sign, and they are used to mark each stop on a line.

Both marker types are clickable. When clicked, the marker reveals a snippet, or a text box, providing the user with more information. The stop marker snippet contains information on the stops name, and which vehicles are arriving next. Vehicle markers reveal which line it is, its final destination and its next stop.

4.5.5 Flow Chart

Figure 4.13 describes the interaction flow of TransitVision. The map screen is the first screen presented at application launch, and is the top left picture. From this screen, it is possible to either search for a line to display on the map, or to change to the Oracle.

The arrows used to depict the flow are color coded, meaning the arrows with the same color are part of the same action. Red and yellow arrows show how to switch between the map and the Oracle. Green is the line search action, and blue is the action for asking the Oracle.



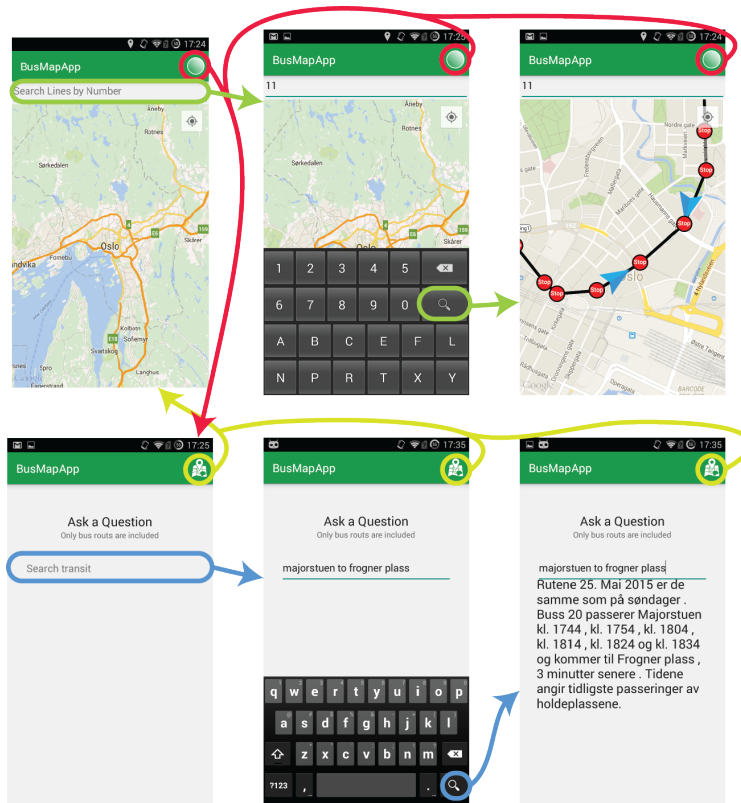


Figure 4.13: Flow Chart of TransitVision

4.6 Implementation of TransitVision

The final product of TransitVision consists of two main parts: an API and a mobile application communicating with the API. This section presents the final result of TransitVision based on previous sections in this chapter.

4.6.1 Mobile Application

The final mobile application consists of two screens, represented as fragments: a Map Fragment and an Oracle Fragment. Each fragment has their individual functionality, and is completely separate.

The list of stops had to be discarded due to the time frame being too short to include it all. Because of this, the final product does not include the functionality



to see a given stop and its future incoming vehicles.

Because of the stop fragment disappearing, the navigation drawer was unnecessary. It is more applicable for larger applications requiring the user to switch between more screens. Since TransitVision only include two screens, the navigation drawer was changed. The screens are now represented as menu items in the action bar. As seen in Figure 4.14, the Oracle can be reached from the icon in the top right corner. Equally, the map can be reached from the top right corner from the Oracle screen as well.

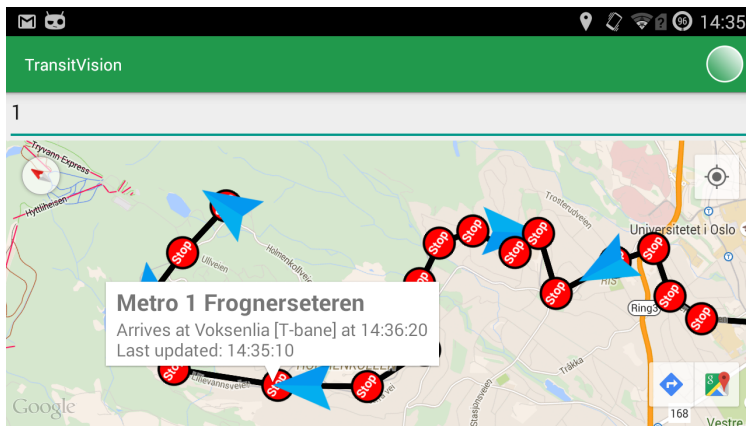


Figure 4.14: The Final Map Screen

Map Fragment

The map screen looks like the digital prototype from Section 4.5.2. Exceptions are 1) the removal of the Navigation Drawer, 2) there is a new menu item in the action bar and 3) the search field appears between the map and the action bar.

The fragment includes a map, and a search field connected to a custom keyboard. Figure 4.14 shows an example of the screen after a user has searched for line number 1. By clicking the search field, a custom keyboard appears. Typing the desired number and searching, will display the desired line on the Google Map. This includes the polyline showing the links between stops, the stops themselves, and the vehicles. Clicking a marker, either stop or vehicle, displays information about the respective marker.

Oracle Fragment

As mentioned in Section 2.1, this master thesis is part of the FUIROS project. Previous projects have focused on the Bus Oracle and schedules, while this thesis target real-time data. Including a fragment for the Oracle, was an effort to combine the two. One of FUIROS newest additions, OsloTUC, is the expansion of BusTUC to Oslo. This new feature created an incentive to utilize the potential of BusTUC within TransitVision.

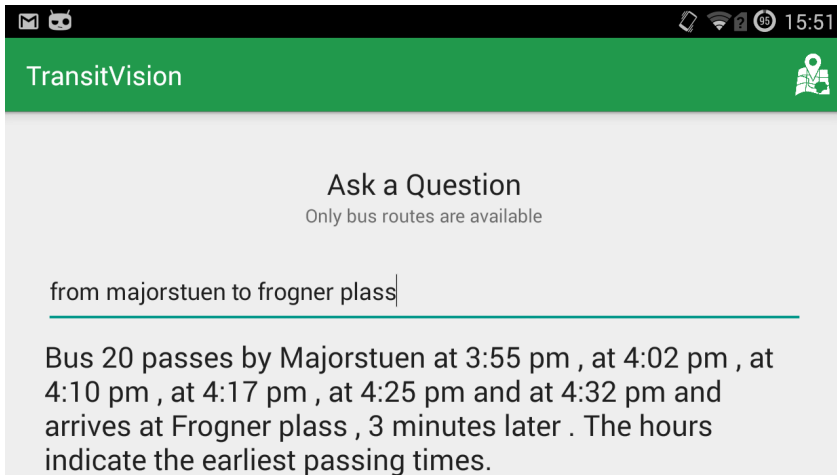


Figure 4.15: Oracle Example

The Oracle screen contains a simple search field. When a user interacts with the search field, he is presented with the default soft keyboard, containing a submit button. The Oracle presents its answer underneath the search field, as shown in Figure 4.15.

Interaction with the OsloTUC (Jacobsson, 2015) is done by sending a POST request to `http://vm-6114.idi.ntnu.no:9001/search`. Requests are formatted as a simple JSON-object similar to `{ "query": "How do I get from Majorstuen to frogner plass?" }`. OsloTUC will then respond with a JSON-object containing a list of arrivals, together with a HTML encoded BusTUC response (Listing 5). TransitVision does not use the HTML tags, and will trim them away before displaying the result.

4.6.2 Providing Vehicle Positions

As mentioned in Section 4.5.1, the process of providing positions is split into several steps. These steps involve data gathering, analysis, calculation and trans-



```

Bus <strong>20</strong> passes by Majorstuen
at <strong>2</strong>:<strong>17</strong> pm ,
at <strong>2</strong>:<strong>25</strong> pm ,
at <strong>2</strong>:<strong>32</strong> pm ,
at <strong>2</strong>:<strong>40</strong> pm ,
at <strong>2</strong>:<strong>47</strong> pm and
at <strong>2</strong>:<strong>55</strong> pm
<br/>and arrives at Frogner plass ,
<strong>3</strong> minutes later .<br/><br/>
The hours indicate the earliest passing times. <br/>

```

Listing 5: Response From OsloTUC

mission. How TransitVision executes these steps is detailed below, where the most important steps are given their individual sections.

The API will start by gathering as much information it can, about the stops and lines of a given transit agency. Afterwards, the API will wait for a position request from the TransitVision application. When such a request is received, it will attempt to estimate the positions of every vehicle on a line.

Updating Stop and Line information

The API requires a lot of information about the operation of vehicles in order to estimate their position. Stop positions form the base of the estimation, together with information about which stops serve which lines. Ruter's online API provides easy access to this data. However, some obstacles must be overcome to use these data for approximating positions. Firstly, the coordinate system used by Ruter is UTM32V, has to be converted to latitude and longitude, to be used by Google Maps. Other operators may use different coordinate systems, and converting them all to latitude and longitude would certainly be preferable. Secondly, the list of stops served by a line is somewhat unordered, and does not contain any information about travel times between the stops. Therefore the order of stops, and time spent between them, is calculated using the real-time information in the *Analyzing Real-Time Data* step.

Querying for Real-Time Arrival Information

The main function of Ruter's real-time system is to provide developers and passengers with real-time arrival estimates for a given stop. These arrival estimates are made using GPS-receivers fitted on vehicles, and logs of previous passes on the same line.

When querying a stop for its next arrivals, a varying number of JSON objects,



similar to the one in Listing 1, are returned. They contain information about arrival time, vehicle, line, destination, and origin. By combining this information from all stops on a given line, it is possible to extract the IDs of vehicles currently in traffic on this line by simply gathering the *VehicleRef* values. It is also possible to compile a list of stops yet to visit for each vehicle, with their respective arrival times. Sorting this list by arrival time will produce the correct order of the stops on the line.

Analyzing the Real-Time Data

Once the API has gathered the real-time data, it must extract the data needed for position estimation. Getting the order of stops and the average travel time between them, is imperative to the position estimation. This is due to Ruter's API not sending any information about when vehicles passed stops in the past, but only its future predictions. In order to position a vehicle, the API must know which stop it passed last, and when. Why this is important is detailed in the next step.

Finding travel time between stops is accomplished by using the list of stop visits received in the previous step, which is sorted by *VehicleRef* and *ExpectedArrivalTime*. By examining the list, it is possible to use the differences in arrival times between stops, to find the average expected travel time between stops. Vehicles yet to start on their route, will often list all the stop visits on the line, and are very useful in this process. After finding all travel times, the API will store them all in a database for future use. Whenever a query is executed, the database will be checked for discrepancies, and updated.

Position Estimation

As the core functionality of the developed system is to provide position estimates, their accuracy and reliability is crucial to the success of the system. Position estimation is done as shown in Figure 4.16, by using the positions of stops and vehicles' estimated arrival times. In this example, vehicle *100001* has just passed a stop, and is no longer listed there. The system does not know the exact time the vehicle visited the previous stop, but it knows when it will arrive at the next stop. It also knows how much time vehicle *100002* expects to spend traversing the same distance. Thereby it can approximate the time vehicle *100001* passed the last stop by simply subtracting the travel time from the expected arrival time on the next stop. In this example, the vehicle likely passed the previous stop at about 12:27:45, as the travel time between the stops is about 2 minutes.

Equation 4.1 is used to calculate the positions. Here pos_1 and t_1 indicate position and time of visit for the next stop. pos_0 and t_0 are position and time of visit for the previous stop. t_x is the current time, and pos_x is the resulting



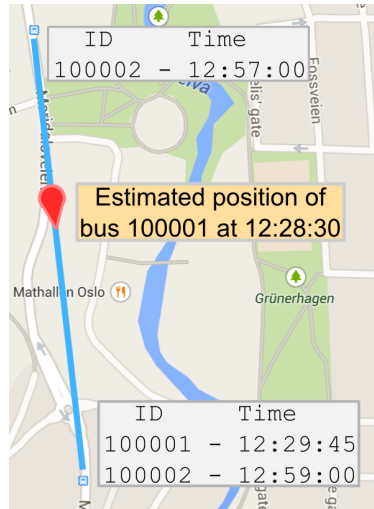


Figure 4.16: Position Estimation Model

position estimate of the vehicle. This method does not take roads into account, and will simply draw the vehicle somewhere on a direct line between the two stops. Including the Google Directions API (Section 2.4.2) in the $(pos_1 - pos_0)$ factor, could allow vehicles to be drawn on the road, for a more realistic look.

$$\text{Given: } t_0 < t_x < t_1 \quad pos_x = \frac{t_x - t_0}{t_1 - t_0} \times (pos_1 - pos_0) + pos_0 \quad (4.1)$$

Once all vehicles on a line have had their positions calculated, they are returned to the user who initiated the request. This estimation method only applies to SIRI-SM systems, as SIRI-VM already include vehicle locations.

Returning the Position Estimates

After the API has estimated the positions of the vehicles on the line, it will return them to the user whom issued the request. This is done by sending a list JSON-objects similar to the one in Listing 6, where each object represents a vehicle. Only the *VehicleID*, *TimeSinceLast*, *Position* and *Bearing* are directly related to the drawing of the vehicle-marker. *ExpectedArrivalTime*, *NextBusStopName*, *DestinationName*, *LineName* and *Transportation* are used to display more information about the vehicle when its marker is clicked. *NextBusStopID* and *DestinationBusStopID* are used for updating the arrival information.



```
{
  VehicleID: "21",
  LineID: "1",
  LineName: "1",
  NextBusStopName: "Haugerud [T-bane]",
  NextBusStopID: 3011530,
  NextBusStopArrival: "2015-04-16T16:54:15+02:00",
  ExpectedArrivalTime: "2015-04-16T16:54:15+02:00",
  ExpectedArrivalTimeMS: 1429196055000,
  Position: {
    Longitude: 10.851413775636043,
    Latitude: 59.92025840288001
  },
  DestinationName: "Ellingsrudaasen",
  DestinationBusStopID: 3011630,
  DestinationArrival: "null",
  PreviousStopID: 3011520,
  TimeSinceLast: 134000,
  Transportation: 8,
  Bearing: 37.450087647845805
}
```

Listing 6: Position of a Subway on Ruter's Line 1

4.7 SIRI Interoperability

The SIRI standard only defines what data to transmit, not how and in which format. Every agency presents a different set of functions to interact with their systems, which may cause interoperability issues. Network limitations and heavy use of local standards, can also become a problem. This section defines the different issues, and proposes a set of properties that a SIRI implementation must have to be compatible with TransitVision.

4.7.1 Server Limitations

Testing during development revealed that Ruter limits the traffic to individual machines. Because of this, the SOA structure became unscalable, as every instance of the application send requests through the same machine. Position queries may also take several seconds to gather enough information because of these limitations. Because of time constraints, TransitVision kept the current prototype structure. When further developing the system, individual phones should communicate directly with Ruter, and other SIRI providers.

To measure the limitations, a simple test was executed using two machines at different locations. Each machine sent a new request as soon as their previous



request was answered. See Section 3.4 for more details about this process.

Table 4.6: Stop Visit Requests from Multiple Machines to Ruter’s API

	Stop 1		Stop 2	
	Alone	Simultaneous	Alone	Simultaneous
Response Size	~ 19 kB	~ 19 kB	~ 207 kB	~ 207 kB
Machine 1	110	113	10	11
Machine 2	112	115	11	11

Based on the results displayed in Table 4.6, it is possible to calculate the upper traffic limit of Ruter’s API. Because *Stop 2* is a transportation hub, it returns a large number of arrivals compared to *Stop 1*. In the table, *Response Size* represents the size of the packages returned from Ruter.

Machine 1 is located in Trondheim, Norway, while Machine 2 is located in Amsterdam, Netherlands. As the machines achieve roughly the same results independently of their location, it seems like Ruter does not treat foreign IPs differently. They also seem to have no impact on the other machine’s response time. Because of the proportionality between Response Size and the amount of responses received, it became apparent Ruter limited the speed, and not the number of responses.

Calculating the upper request limit was done by using the mean number of requests per stop. For instance, the mean amount of requests was $(110 + 113 + 112 + 115)/4 = 112.5$, for *Stop 1*. This gave $\frac{19kB \times 112.5}{60s} = 35.6kB/s$ for *Stop 1* and $\frac{207kB \times 11}{60s} = 38.0kB/s$ for *Stop 2*, indicating that Ruter’s limit is close to $40kB/s$ for each machine.

Table 4.7: Network Traffic Limitations for Kolumbus’ API

Response Size	Requests per minute	Average speed
3557 Bytes	730	$43kB/s$
913 Bytes	1076	$16.4kB/s$
717 Bytes	1078	$12.9kB/s$
3279 Bytes	751	$41kB/s$
2033 Bytes	1081	$36.62kB/s$

Pinpointing Kolumbus’ traffic limitations was done using the method described in Section 3.4. The only exception being that the test only used one machine. Table 4.7 shows the results, which indicate that Kolumbus limit traffic



at about 18 requests per second, and $45^{kB}/s$. Because of the decreased response size, Kolumbus' API can send packages about ten times faster than Ruter, with the same traffic limit. Consequently, Kolumbus could potentially create a more responsive experience than Ruter.

4.7.2 Prerequisites for the SIRI-SM Implementations

Many SIRI-SM systems use different versions of SIRI with slight deviations from the standard. Below is a list of prerequisites a SIRI-SM implementation must meet before TransitVision can use it. Data, such as list of lines and stops, may be available through different formats like GTFS or REGTOPP. If the SIRI-SM implementation fails on one of the following prerequisites, TransitVision is unable to estimate positions for that agency.

Fetching arrival times At the core of SIRI-SM is the functionality to get information about incoming vehicles at a given stop. SIRI-SM is useless without arrival times.

Distinguishing between vehicles using an ID It is imperative that the system can distinguish between individual vehicles. SIRI defines this ID as *VehicleRef* (see Listing 1 in Section 2.3.1), but it is not always included in the SIRI-SM data.

Access to complete list of stops Usually a stop ID has to be presented to query for real-time arrivals. The positions of the stops are also a crucial part of the positioning algorithm described in Section 4.6.2.

Access to complete list of lines, and their stops When querying for a specific line, it is important that TransitVision knows which stops serve that line.

Network traffic limitations Limits on networking traffic can cause a severe performance hit to the map, shown in Section 4.7.1, a $40^{kB}/s$ cap is just barely enough.

4.7.3 Prerequisites for the SIRI-VM Implementations

As SIRI-VM already has the positions available, TransitVision does not have many prerequisites for SIRI-VM implementations. SIRI-VM systems usually send a list of all vehicles in transit through a single web query (similar to the Tampere example in Appendix B). TransitVision requires no further information from these systems. While not required, it is preferable to have a list of stops, so they can be drawn on the map together with the vehicles.



Other SIRI-VM implementations return vehicle information on a line by line basis. For those implementations TransitVision must have a list of the available lines. Both line information and stop information may be available through GTFS, REGTOPP or a similar format.

4.7.4 Compatibility of Different SIRI Implementations

To establish the interoperability of TransitVision, numerous SIRI implementations were evaluated. Figure 4.8 shows the results from this examination. Agencies included are Ruter¹⁷, Kolumbus¹⁸, Fram¹⁹, and AtB²⁰ in Norway, Metropolitan Transportation Authority (MTA)²¹ in USA, and Tampere Public Transport²² in Finland. The test examined the various prerequisites within each of the implementations. AtB's system is currently in development, and could not be examined. It is included because it is a prime target for expansion as it is located in the same city as NTNU.

Table 4.8: Comparison of SIRI Implementations

Agency	SIRI Solutions	Format	Coordinate System	Eligible
Ruter	SM	JSON & XML	UTM32V	Yes
AtB	SM is in development	XML	UTM32V	Unknown
Kolumbus	SM	JSON & XML	Lat/Long	Yes with GTFS data
MTA	SM & VM	JSON & XML	Lat/Long	Yes
Tampere Public Transport	SM & VM	JSON & XML	Lat/Long	Yes
Fram (Ferries)	SM & VM	XML	Lat/Long	Yes

¹⁷<http://labs.ruter.no/ofte-stilte-spoersmaal.aspx>

¹⁸<http://next.kolumbus.no/2013/08/30/forste-sniktitt-pa-sanntids-api-et>

¹⁹<http://www.frammr.no/info/Sanntidsinformasjon>

²⁰<https://www.atb.no/aapne-data/category419.html>

²¹<http://bustime.mta.info/wiki/Developers/SIRIIntro>

²²<http://developer.publictransport.tampere.fi/pages/en/siri.php>



4.7.5 Expanding to Tampere

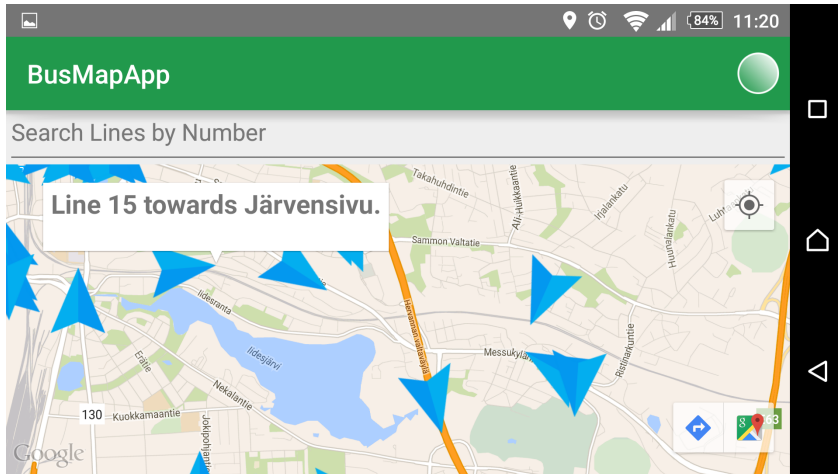


Figure 4.17: TransitVision using Tampere SIRI-VM Data

An attempt of including Tampere’s SIRI-VM implementation was successfully made. Their system provides locations of all buses using a single query, which makes displaying the vehicles very straight forward. It must be noted that it was a simple interoperability test, and Tampere is not currently available in the application. This is mainly because it draws all buses in Tampere, and phones have a tendency to lag when drawing and updating hundreds of markers at the same time. The Tampere test version can be seen in Figure 4.17, and an example for what is included when Tampere Public Transport sends their vehicle information can be found in Appendix B.

4.7.6 Prospect of including Kolumbus’ API

Kolumbus’ API uses two simple URLs: one for sending stop information, and one for real-time arrival estimates. Unfortunately, there is no line information in the API, this can, however, be obtained through Kolumbus’ detailed GTFS data.

There are some differences in the data sent by Kolumbus compared to Ruter. Kolumbus has decided to omit SIRI data they have deemed unnecessary, which explains the extreme difference in response size when compared to Ruter (Section 4.7.1). Even with these limitations, Kolumbus still meet the requirements set in Section 4.7.2. Ruter send their date-times in ISO 8601 format (International Organization for Standardization, 1988), while Kolumbus send their date-times



in milliseconds since Unix epoch²³. Ruter follows the Norwegian UTM32V coordinate system, which has to be converted to latitude and longitude to be used in Google Maps. Kolumbus provide their coordinates in latitude and longitude, thus no conversion is necessary.

Including Kolumbus in TransitVision should be easy once information about the lines can be gathered. Lack of line information is the main obstacle when including Kolumbus data. It may be overcome by creating a GTFS parser, to gather the missing line information. Such a GTFS parser could also be used to include many other SIRI implementations.

4.8 User Testing

After completing TransitVision, the test phase could begin as described in Section 3.5. 15 participants answered a SUS questionnaire after doing a short test of the application. The SUS questionnaire resulted in a score of 86. Participants were also asked to give their thoughts on TransitVision, which gave valuable feedback.

4.8.1 Feedback Received from User Testing

Testers discovered several flaws, and potential improvements for the application. Some participants noted that the application does not notify the user if the API is unavailable. Instead, it might seem like the application is searching for the line as expected.

When a request is sent, there is a little response delay, due to the server needing to gather all data to calculate vehicle positions. Because of the response delay, the user is sometimes left wondering whether the application is working as intended. To combat this, the participants suggested some sort of loading screen, to indicate TransitVision is processing the request.

Other participants wanted an easier way of finding the name of a stop. They suggested having the name of the stops appear when zooming in on the map. Thereby eliminating the need for the user to tap on the marker to see the name of the stop. By adding this feature, the user would only have to tap the stop marker to see when the next vehicles arrive.

One of the participants also suggested different vehicle icons for the different modes of transport. This removes the need to tap vehicle markers in order to find its vehicle type. The first icons indicated the vehicle type, but because of earlier feedback claiming the icons looked like stops, they were discarded (further discussed in Section 4.5.4).

²³00:00:00 UTC, Thursday, January 1st 1970

4.8.2 The SUS Score

The survey and testing phase of the project was conducted with 15 participants. Section 2.7 describes the test, and justifies the number of participants. Most of the participants lived in Oslo, and had used other public transit mobile applications before.

Table 4.10: SUS Results

Participant #	1	2	3	4	5	6	7	8	9	10
1	1	1	3	1	4	2	5	2	5	1
2	5	1	4	1	4	1	5	1	5	2
3	2	1	5	1	1	1	5	1	5	1
4	4	1	5	1	4	1	4	1	3	1
5	5	2	4	1	4	5	1	1	2	1
6	4	1	5	2	4	1	4	2	5	1
7	4	1	5	2	4	2	5	1	4	1
8	5	1	5	1	5	1	5	1	5	1
9	4	2	4	1	4	1	5	3	4	2
10	5	1	5	1	5	1	4	2	4	2
11	2	1	3	1	4	2	4	2	3	1
12	4	1	5	1	4	1	4	1	5	2
13	5	2	5	1	5	1	5	1	5	1
14	5	2	4	1	4	1	5	2	4	1
15	4	1	5	2	5	2	5	1	4	2

Table 4.10 lists the responses received from the testers through the SUS questionnaire, further described in Section 2.7. Each row in the table indicates a new participant, while the columns represent the statements. The statements are listed in Appendix A.1.

By using the equations 2.1, 2.2 and 2.3 from Section 2.7.1, the SUS score of TransitVision could be computed. After shifting all responses to range from 0 to 4 by using Equation 2.1, the mean response of each question computes to 2.93, 3.73, 3.47, 3.80, 3.07, 3.47, 3.40, 3.53, 3.20 and 3.67. Adding them together and multiplying by 2.5, results in a SUS score of $85.67 \approx 86$. Which, according to Bangor et al. (2008), is close to being considered superior, and confirms the TransitVision’s usability as described in Section 4.4.

Table 4.11 shows statistics about the ten statements, based on the fifteen responses from Table 4.10. It includes the mean values, the standard deviation, a confidence interval with a 95% confidence level, and the range. The true range column contains the range values rounded to integers, making them more suitable



as SUS responses.

The least desirable reactions are lower numbers for odd numbered statements, and higher numbers for even numbered statements. Picking out the worst possible responses within the true range in Table 4.11, gives 3, 1, 4, 1, 4, 2, 4, 2, 4, 2. Shifting the numbers to range from 0 to 4 with the method earlier mentioned in Section 2.7, results in 2, 4, 3, 4, 3, 3, 3, 3, 3, 3. Adding them together and multiplying by 2.5 gives:

$$(2 + 4 + 3 + 4 + 3 + 3 + 3 + 3 + 3 + 3) * \frac{5}{2} = 77.5 \quad (4.2)$$

The result from the worst possible response within the range is still a good score at 77.5 (Bangor et al., 2008).

Table 4.11: Statement Statistics

Statement	Mean	Standard Deviation	Confidence Interval	Range	True Range
1	3.93	1.2365	±0.63	3.30 - 4.56	3 - 5
2	1.27	0.4422	±0.22	1.05 - 1.49	1 - 1
3	4.47	0.7180	±0.36	4.11 - 4.83	4 - 5
4	1.20	0.4000	±0.20	1.00 - 1.40	1 - 1
5	4.07	0.9286	±0.47	3.60 - 4.54	4 - 5
6	1.53	1.0242	±0.52	1.01 - 2.05	1 - 2
7	4.40	1.0198	±0.52	3.88 - 4.92	4 - 5
8	1.47	0.6182	±0.31	1.16 - 1.78	1 - 2
9	4.20	0.9092	±0.46	3.74 - 4.66	4 - 5
10	1.33	0.4714	±0.24	1.09 - 1.57	1 - 2

In order to show the variations in the responses better, an error bar diagram was created for a visual representation. The diagram is displayed in Figure A.1 in Appendix A.2.



Chapter 5

Discussion

Initial plans were determined during the autumn project, such as platform and real-time standard to use. Thus, the decisions on which features to include and how to organize the project were established early. Because of the emergence of several ambitious ideas, the project quickly became too big to fit within the time constraint. Eventually, the scope of the application required a reduction to make the map feature as good as possible.

TransitVision turned out close to the original plan. The map functionality works well on a smartphone, and does not need a computer-sized screen to be functional. During development, it was discovered that simultaneously drawing markers for all stops and all vehicles in Oslo would be too much work for a regular smartphone to handle. This would also look very messy on a small smartphone screen, and be counterproductive by giving the user too much information at the same time. Because of this, the application requires input from the user to show a desired line. Even though there is some latency when displaying vehicles, as described in Section 4.8.1, the authors are pleased with how fast the application is calculating vehicle locations based on the real-time data retrieved from the server.

The mobile application uses a simple design, and thereby emphasizes the map functionality. UI design is also true to the standard concepts for Android application development. There are not introduced any new elements in the application which users may be unfamiliar with. The authors believe TransitVision promotes an easy flow of information on user request, and does not push unnecessary data or details.

The discoveries made during development, an analysis of the results, and discussion are included in this chapter.



5.1 Preliminary Study

Information gathered through the preliminary survey helped in the development of TransitVision. The section of the survey where participants answered questions about the transit application they used made it possible to map functionality that worked well.

Information on waiting time helped with answering Research Question 3, and made the authors realize that mobile applications may not help that much with the time spent waiting for a bus. Given that the waiting time data gathered only captures the participants' perceived waiting time, it might be possible to point out that mobile application users expect more accurate data. When the bus does not arrive when it says it should, the mobile application users get frustrated. Travelers using time tables are aware the bus is not necessarily on schedule, and thus do not mind waiting. Based on the feedback, it might seem like the existing mobile applications do not present real-time data in a satisfactory manner.

5.2 Fulfillment of Requirements

Even though several system requirements were created early in the development, not all of them were feasible mostly due to the time constraint. The requirements not implemented are presented and explained in this section. As a reminder, the requirements are listed in Table 4.4 in Section 4.2.

Table 5.1: Status of Requirements

Status	Requirements
Fully Completed	M1, M2, M5, M10, M11, S1, S2, S3, S4, S5, S7, S9, S6
Partly Completed	M3, M4, S11
Not Completed	M6, M7, M8, M9, S8, S10

Table 5.1 shows an overview of the requirement fulfillment. Requirement M3 and M4 are only partly implemented as originally intended. To display stops (M3) the user has to search for a line, and only the stops managed by that line will be displayed. The map can draw all stops, just not at the same time. Requirement M4 was initially related to the list of stops, where the user could search for a stop and see incoming vehicles. By searching for a line, the related stop markers are clickable, making the user able to see incoming vehicles.

Requirement S10 was one of the lowest rated server requirements, and was quickly deemed unnecessary. S11 was only partially implemented and is discussed in detail in Section 5.3.3. The lack of S8 has a severe impact on the interoper-



ability of TransitVision. Many SIRI systems use JSON because it is more human readable than XML, and is usually smaller in size. SIRI is primarily an XML standard, and TransitVision should be able to read XML files to become interoperable. Because development was done in node.js and JavaScript, it was easier to use JSON. Switching to XML should be manageable as the SIRI-XML files contain the same data and attribute names as the SIRI-JSON objects. Node.js packages like `xml2js`¹ can help making the process easy.

5.3 Obstacles Found During Development

As with any IT system, many issues arose during development. Some of these altered the structure of the implementation, and are described below.

5.3.1 Vehicle Animations

Initially the vehicles would be static on the map, and would simply display their locations at the time of request. Ideally, the application could retrieve the positions from the API regularly, creating a smooth animation. However, because of the limitations of the servers, this seems to be problematic. To solve this issue, the API sends a list of estimated stop visits to the mobile application, instead of sending position estimates. This means that the individual phones would handle the position estimation, reducing the API to a simple adapter between the different SIRI implementations and the application.

The application calculates the position estimates every frame (frame rate may vary depending on the device running the application), creating smooth vehicle animations. The frequency of the real-time data updates increases the closer a vehicle is to the stop, spanning from 12 to 60 times an hour. Once a vehicle passes a stop, TransitVision will instantly update the vehicle's next stop, further ensuring the accuracy of the position estimate.

All these precautions for updating frequency ensure accurate position estimates, smooth animations, and reduced Internet traffic. The phone estimates the positions using the same technique as described in Section 4.6.2. Server-side estimates are still available, but the application does no longer use them.

5.3.2 Irregularities in Ruter's Data

When asking Ruter for information about a given line, it will present a list of stop IDs served by this line. The order of the list is usually correct, but there are several problems associated with it.

¹<https://www.npmjs.com/package/xml2js>



1. Stops visited while going in one direction, are not always the same as the opposite direction. For instance, a vehicle sometimes serves a stop in only one direction. Ruter's metro stop *Gulleråsen* on line 1, is an example of this.
2. Some lines visit different stops on their last and first run of the day. Trains, metros and trams sometimes park at a depot not directly connected to their line. Consequently, most metro lines list *Ryen*, and stops in between as a part of their line, because *Ryen* is the largest depot in Oslo.
3. Some of the listed stops are not visited by that line number. For example line 6 lists stops visited by line 4, as metros on these two lines will switch from line 6 to line 4, at a certain stop.

TransitVision handles the first issue in the Analyzing the Real-Time Data step in Section 4.6.2. TransitVision deduces the correct order of stops by comparing the real-time data with the line data. TransitVision has issues number 2, but the rarity of the case makes the issue negligible. Issue 3 causes TransitVision to draw line 6 with the stops of line 4. It will not draw any vehicles between those stops, as line 6 has no arrivals there. Similar issues may exist for other lines.

5.3.3 Google Directions API

As suggested in Section 4.6.2, TransitVision should draw buses on roads in the Google Map. The Google Directions API (Section 2.4.2) is able to do just that, and it is compatible with Google Maps.

There is currently an unused Google Directions integration in the developed web API. The integration was discarded mainly because of inaccuracies in Ruter's stop positioning. Ruter uses a single coordinate to denote stops, even though there usually are stops on both sides of the road. Ruter has simply chosen one of the stops as the coordinate for both. Consequently, Google Directions would sometimes produce an extreme detour to get to the other side of the road, especially if the stop is on the highway.

Another issue with the Google Directions API is that it is not made for public transportation in the sense that it will never present a route that follow a bus-only road. It will instead take yet another detour. Manually editing the bus stop location slightly or altering the polyline drawn on the map can solve both these issues. The large amount of such errors made them too time consuming to fix.

Because Google Directions API set a limit on the amount of queries an application can send in a day, the API will store the directions between two stops. By combining the directions, TransitVision can draw a polyline following the entire route. By storing the directions, it is possible to include Google Directions in a future version.



5.4 User Test

The results from SUS were very promising, giving TransitVision a score of 86. Even though this is not a superior score, it is a decent start. Based on this, the application could expand features, and then run another SUS questionnaire. The new score could be compared to the previous to see the change in reaction, and used to evaluate whether the responses are positive or negative to the new feature. An example of a new testable feature could be connecting the Oracle with the map, displaying the query result on the map. This idea is further explained in Section 6.1.1.

There are some irregularities in the responses, as seen in Table 4.10 in Section 4.8.2. An example is participant 5's response to statement 7: *I would imagine that most people would learn to use this system very quickly*. The rest of the responses to statement 7 are only 4s and 5s. Statement 4: *I think that I would need the support of a technical person to be able to use this system*, should have been answered differently by participant 5, due to the response to number 7. Because if most people would take a long time learning the system, they should need support from technical persons to be able to use it. It is impossible to know why such irregularities occur. It may stem from the participant 1) not understanding the statement, 2) reading the statement incorrectly (for instance thinking that statement 7 said *wouldn't*, instead of *would*), or 3) simply not paying attention to what they were responding to. Given that the participant is not agreeing with himself, he could have been disregarded from the sample. However, the complete sample was included in the calculation of the SUS score.

The number of participants was argued in Section 3.5. However, a larger sample size would have made the result even better. With a sample size of 15, each individual response have a big effect on the final score. With the irregularities discovered, the final score might not be entirely correct.

While SUS is a quantitative measure, the study could also benefit from qualitative feedback from testers. Qualitative feedback is easier to interpret and use when improving a user interface. In addition to SUS, some testers gave verbal feedback about TransitVision. In retrospect, more verbal feedback would be preferable from the user test, combined with response from more users.

5.5 Research Questions and Goal Achievements

The following section discusses in which degree the goal has been fulfilled, and the research questions answered.



5.5.1 The Goal

Develop a smartphone map application, which approximates the locations of public transportation vehicles using SIRI-SM real-time arrival data. TransitVision fulfills this goal. The final result is a mobile application capable of estimating vehicle positions based on real-time data gathered from Ruter's API. Currently it supports SIRI-SM, but was also tested for SIRI-VM, which was easily included in the system.

5.5.2 Research Question 1

How can the SIRI standard be utilized to develop an application that can be applied to any SIRI implementation, without considerable modifications? SIRI is not a very strict standard, and every implementation of it is different regarding function names, traffic limitations, file-format, or data-format. Because of this, not all SIRI implementations are applicable. For a SIRI-SM implementation to be classified as usable for TransitVision, it must fulfill the prerequisites set in Section 4.7.2. While these prerequisites apply to TransitVision, other applications utilizing the SIRI standard may have different requirements.

5.5.3 Research Question 2

How can a mobile map application, using SIRI-SM, provide a satisfactory representation of real-time data? TransitVision is a possible solution to this question. The results from the questionnaire deemed a positive response from the testers with a SUS score of 86. This shows that TransitVision delivers a satisfactory representation of the real-time data using SIRI-SM on a map.

5.5.4 Research Question 3

What impact does mobile transit applications have on travellers frustration and perceived waiting time? Numerous applications and their usage were examined in the preliminary study. However, the survey indicated the applications had little to no impact on the participants' perceived waiting time, compared to participants of more traditional methods. Thus, TransitVision focuses on a different way of presenting real-time data, which the testers had a positive response to. With some more work and perfecting, TransitVision might have a positive impact on waiting time.



Chapter 6

Conclusions

As travelers using transit applications think they wait about as long as other travelers, these applications are perhaps inadequate for presenting real-time transit data. TransitVision provides a working alternative, and can be converted to almost any SIRI-SM system with few code modifications. A user test confirmed TransitVision's usefulness and provided ideas for future improvements.

SIRI is a very flexible standard, and every implementation of it is different. SIRI does not foster interoperability to a satisfactory degree, because agencies use different data formats. To achieve interoperability, third party applications must be tailored for the individual SIRI implementations.

Usability is an important factor in transit applications, and is often the deciding factor when choosing which application to use. By expanding TransitVision to include features from similar applications, while maintaining its current level of usability, it should be able to challenge its competition. Because of its unrivaled map functionality, it may even be able to outmatch them.

TransitVision shows that it is possible to generate position approximations for public transportation vehicles using real-time arrival estimates. Hopefully, more widespread use of such applications might give public transportation agencies more incentive to make their GPS data publicly available. It may also have a positive impact on the travelers' satisfaction.

Through this thesis, TransitVision became a successful proof-of-concept, but still requires work before it can become a finished product.



6.1 Future Work

Several ideas for future work with TransitVision have been discussed, and this section presents two of these suggestions.

6.1.1 Connecting the Oracle with the Map

The finished version of TransitVision's two main features (the map and the Oracle) are completely separate, and define two completely different ways of presenting arrival data. By connecting the Oracle and the map it would be possible to draw the response from the Oracle on the map, including the vehicle's position. The trip can be drawn on the roads using the Google Directions API. TransitVision will require modifications to the Google Directions response, as discussed in Section 5.3.3. By implementing this Oracle improvement, it will be better integrated into the map application.

6.1.2 Expand to Other Cities

The primary incentive to choose SIRI, was the aspect of interoperability between multiple cities. Several approaches have been discussed, but the focus of this project has been on developing an adaptable prototype application and not deploying it on as many systems as possible. To reach as many cities as possible, TransitVision could also incorporate other real-time standards like NextBus (Section 2.3.2), or GTFS-realtime (Section 2.3.3).

AtB in Trondheim has promised the release of their SIRI-SM implementation later this year, and because the FUIROS project is centered in Trondheim, it is the ideal candidate for further expansion.







Acronyms

- AKT** Agder Kollektivtrafikk AS. 6
- API** Application Program Interface. 7–9, 13, 14, 17, 31, 32, 34–36, 39, 44, 47–49, 51, 52, 54, 55, 61, 62, 64, 66, 73, 77, 78
- APTA** American Public Transportation Association. 7
- AVL** Automatic Vehicle Location. 8
- BSON** Binary JSON. 15, 35
- BusTUC** Bus, The Understanding Computer. 1, 5, 31, 32, 46
- CEN** European Committee for Standardization. 2, 7, 8
- DDOS** Distributed Denial of Service. 23
- FUIROS** the Futures Ultimate Intelligent Route-Organizing System. 1, 5, 46, 66
- GPL** General Public License. 7
- GPS** Global Positioning System. 1, 43, 47, 65
- GTFS** General Transit Feed Specification. 6, 9, 11, 15, 36, 52–55, 66
- JSON** JavaScript Object Notation. 9, 11, 14, 15, 32, 34, 35, 46, 47, 49, 53, 61
- MBTA** Massachusetts Bay Transportation Authority. 11
- MTA** Metropolitan Transportation Authority. 53



- MultiBRIS** Multiple-platform approach to the Ultimate Bus Route Information System. 5
- NeTEx** Network Timetable Exchange. 7
- NTNU** Norwegian University of Science and Technology. 1, 5, 53
- REGTOPP** Regional Trafikkopplysing. 6, 52, 53
- SIRI** Service Interface for Real Time Information. 2, 4, 7–9, 11, 24, 27, 31, 36, 50, 52–55, 61, 64–66
- SIRI-SM** SIRI Stop Monitoring Service. 2, 9, 10, 23, 32, 49, 52, 64–66
- SIRI-VM** SIRI Vehicle Monitoring Service. 9–11, 17, 49, 52–54, 64
- SOA** Service-Oriented Architecture. 36, 50
- SUS** System Usability Scale. 18, 19, 24, 27, 55–57, 63, 64
- TABuss** Tore Amble Buss. 5
- UTM** Universal Transverse Mercator coordinate system. 34, 47, 53, 55
- XML** EXtensible Markup Language. 7–9, 11, 32, 34, 53, 61



Bibliography

- Amble, T. (2000). BusTUC - A Natural Language Bus Route Oracle. In *Proceedings of the sixth conference on Applied natural language processing*, pages 1–6. Association for Computational Linguistics.
- Andersstuen, R. and Engell, T. B. (2011). MultiBRIS: A Multiple Platform Approach to the Ultimate Bus Route Information System for Mobile Devices. Master’s thesis, Norwegian University of Science and Technology.
- Andersstuen, R. and Marcussen, C. J. (2012). TaleTUC: Automatic Speech Recognition for Bus Route Information System. Master’s thesis, Norwegian University of Science and Technology.
- Antrim, A., Barbeau, S. J., et al. (2013). The many uses of gtfs data—opening the door to transit and multimodal applications. *Location-Aware Information Systems Laboratory at the University of South Florida*.
- Bangor, A., Kortum, P. T., and Miller, J. T. (2008). An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 24(6):574–594.
- Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition.
- Bratseth, J. S. (1997). Bustuc - a natural language bus traffic information system. Master’s thesis, Norwegian University of Science and Technology.
- Brooke, J. (1996). Sus-a quick and dirty usability scale. In Jordan, P. W., Thomas, B., McClelland, I. L., and Weerdmeester, B., editors, *Usability Evaluation in Industry*, chapter 21, pages 189–194. London: Taylor & Francis.
- CEN/TC278/WG3/SG7 (2005). Siri management overview - white paper.
- CEN/TC278/WG3/SG9 (2014). Public transport - Network and Timetable Exchange (NeTEx).



- Charland, A. and Leroux, B. (2011). Mobile application development: web vs. native. *Communications of the ACM*, 54(5):49–53.
- Chasseur, C., Li, Y., and Patel, J. M. (2013). Enabling json document stores in relational systems. In *WebDB*, pages 1–6.
- Couper, M. P. (2000). Review: Web surveys: A review of issues and approaches. *Public opinion quarterly*, pages 464–494.
- Diehl, M. and Stroebe, W. (1987). Productivity loss in brainstorming groups: Toward the solution of a riddle. *Journal of personality and social psychology*, 53(3):497.
- Faulkner, L. (2003). Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3):379–383.
- Finstad, K. (2006). The system usability scale and non-native english speakers. *Journal of Usability Studies*, 1(4):185–188.
- Gibson, R. and Erle, S. (2006). *Google Maps Hacks*. " O'Reilly Media, Inc."
- Goadrich, M. H. and Rogers, M. P. (2011). Smart Smartphone Development: iOS Versus Android. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, pages 607–612. ACM.
- Grisby, D. (2013). Apta surveys transit agencies on providing information and real-time arrivals to customers. *Am. Public Transit Assoc*, pages 1–13.
- International Organization for Standardization (1988). *Data Elements and Interchange Formats: Information Interchange: Epresentation of Dates and Times*. International Organization for Standardization.
- Jacobsson, E. (2015). Oslostuc - natural language bus oracle for a new city. Master's thesis, Norwegian University of Science and Technology.
- Jick, T. D. (1979). Mixing qualitative and quantitative methods: Triangulation in action. *Administrative science quarterly*, pages 602–611.
- Kaner, C. (2003). An introduction to scenario testing. *Software Testing & Quality Engineering magazine*.
- Knowles, N. (2008). *SIRI Handbook & Functional Service Diagrams*. Ki-zoom Limited. <http://user47094.vs.easily.co.uk/siri/schema/1.3/doc/Handbook/Handbookv15.pdf>.



- Lau, F., Rubin, S. H., Smith, M. H., and Trajković, L. (2000). Distributed denial of service attacks. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 3, pages 2275–2280. IEEE.
- Li, X., Liu, Z., and Jifeng, H. (2004). A formal semantics of uml sequence diagram. In *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pages 168–177. IEEE.
- Mahapatra, L. (2013). Android vs. iOS: What’s the Most Popular Mobile Operating System in Your Country? <http://www.ibtimes.com/android-vs-ios-whats-most-popular-mobile-operating-system-your-country-1464892>. Accessed: 2014-10-29.
- Marcussen, C. J. and Eliassen, L. M. (2011). TABuss: An Intelligent Smartphone Application. Master’s thesis, Norwegian University of Science and Technology.
- McHugh, B. (2013). Pioneering open data standards: The gtfs story. *Edited by Brett Goldstein with Lauren Dyson*, pages 125–135.
- Nettbuss (2014). Årsrapport 2013, nettbuss. <http://www.nettbuss.no/www/resources/nbno/f/fc02b61a1b120b332dfda8c6cefb82f9.pdf>.
- NextBus Incorporated (2013). Public xml feed. <http://www.nextbus.com/xmlFeedDocs/NextBusXMLFeed.pdf>.
- Oslo og Akershus Trafikkservice AS (1996). Regtoppformatet versjon 1.2. http://labs.trafikanten.no/media/12753/RF_1-2-1-1.pdf.
- Porter, S. R. and Whitcomb, M. E. (2003). The impact of contact type on web survey response rates. *Public Opinion Quarterly*, pages 579–588.
- Schmier, K. and Freda, P. (2002). Public transit vehicle arrival information system. US Patent 6,374,176.
- Sims, C. and Johnson, H. L. (2014). *Scrum: a Breathtakingly Brief and Agile Introduction*. Dymaxicon.
- Snyder, C. (2003). *Paper prototyping: The fast and easy way to design and refine user interfaces*. Newnes.
- Tibaut, A., Kaučič, B., and Rebolj, D. (2012). A standardised approach for sustainable interoperability between public transport passenger information systems. *Computers in Industry*, 63(8):788–798.
- Tilkov, S. and Vinoski, S. (2010). Node.js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, (6):80–83.

BIBLIOGRAPHY

- Tullis, T. S. and Stetson, J. N. (2004). A comparison of questionnaires for assessing website usability. In *Usability Professional Association Conference*, pages 1–12.
- Watkins, K. E., Ferris, B., Borning, A., Rutherford, G. S., and Layton, D. (2011). Where Is My Bus? Impact of Mobile real-Time Information on the Perceived and Actual Wait Time of Transit Riders. *Transportation Research Part A: Policy and Practice*, 45(8):839–848.



Appendix A

SUS

This chapter contains extra information on SUS.

A.1 The Questionnaire

This list contains all statements used in the SUS evaluation.

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.



A.2 Error Bar Diagram

Figure A.1 displays the mean answers to each statement, including the confidence interval at a 95% confidence level.

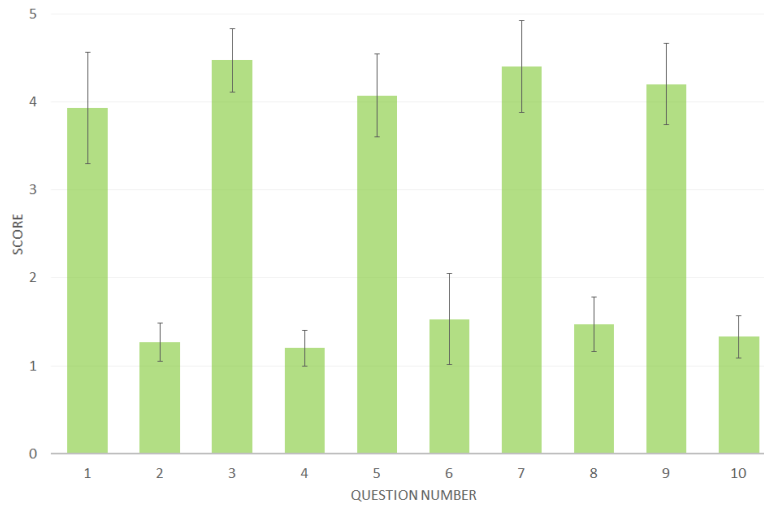


Figure A.1: SUS Answers With 95 % Confidence Level



Appendix B

SIRI-VM Example

```
{
  ValidUntilTime: 1424172074013,
  MonitoredVehicleJourney: {
    LineRef: {
      value: "11"},
    DirectionRef: {
      value: "2"},
    FramedVehicleJourneyRef: {
      DataFrameRef: {
        value: "2015-02-17"},
      DatedVehicleJourneyRef: "1235"},
    OperatorRef: {
      value: "paunu"},
    OriginName: {
      value: "Korvenkatu 44",
      lang: "fi"},
    DestinationName: {
      value: "Sarankulma",
      lang: "fi"},
    Monitored: true,
    VehicleLocation: {
      Longitude: 23.7416098,
      Latitude: 61.45785},
    Bearing: 43,
    Delay: "POYOMODTOHOM33.000S",
    VehicleRef: {
      value: "paunu_136"}
  },
  RecordedAtTime: 1424172044013
},
```

Listing 7: SIRI-VM JSON Returned from Tampere Bus Service Online API





Appendix C

Digital Attachements

C.1 Github Repositories

The QR codes direct you to the repositories of the application. One for the API, <https://github.com/OleKN/busMapAPI>, and one for the mobile application, <https://github.com/sofiabakke/BusMapApp>.



(a) The Web API



(b) The Mobile Application

Figure C.1: Links to Public Code Repositories



C.2 TransitVision Google Play Link

The QR code in Figure C.2 will take you to the Google Play listing for TransitVision. It can alternatively be reached through: <https://play.google.com/store/apps/details?id=no.application.sofia.busmapapp>



Figure C.2: Link TransitVision in Google Play Marketplace



C.3 Video of TransitVision in Action

The QR code below will direct you to a short YouTube video. The link goes to the video <http://youtu.be/GVLsDureRfs> and shows the main features of the map.



Figure C.3: Link to Video Showing TransitVision in Action



Appendix D

Setting Up the Project

This chapter explains the steps required for setting up the API. To use a local API, the address in the mobile application source code has to be changed. This is done by locating line 42 of the class `RouteMarkerHandler`:

```
private final String IP = "http://api.bausk.no/";  
    //Change IP address of server here
```

The first step is to download all the necessary frameworks and code to your computer. The Github repositories for both the mobile application and server are listed in Section C.1. For this walk through, you need to clone the server repository. Next, download Node.js from <https://nodejs.org>, or through a package manager. Follow the installation guide on their website. MongoDB can be downloaded from <https://www.mongodb.org>.

Run npm install Locate the cloned code from the previous step. Open a terminal or command prompt and navigate to the `busMapAPI` folder, which is the root folder of the source code. Type `npm install` and hit execute. This will install all node.js dependencies required to run the API.

Start MongoDB Start MongoDB by executing `service mongod start` in your command prompt/terminal.

Start the API Now start the API in the following way.

1. The first time the API is run it must be run with the `-updateDB` argument to update the database. Following restarts will simply call `node app.js`.
2. To prevent the API from shutting down when logging out, `forever` can be used. It can be installed by using the command `npm install forever -g`. After installing `forever`, the API is started with `forever app.js`. `Forever` will also restart the API if it crashes, so do not include `-updateDB` as it will then update the database if it has to restart.



APPENDIX D. SETTING UP THE PROJECT

After starting the API it can be reached at port 3000. Look inside `app.js` for a complete list of available functions.



Appendix E

Preliminary Study

E.1 Preliminary Study Questionnaire

This list contain the questions posed in the preliminary study questionnaire, described in Section 3.1.

1. How many times a week do you commute by bus?
2. What is your work status? (Voluntary)
3. How old are you? (Voluntary)
4. In which city do you utilize bus the most?
5. In what context do you commute by bus?
6. How long do you usually wait for the bus?
7. Do you think knowing the exact location of the bus would be helpful to reduce the time spent waiting for the bus?
8. On a scale from 1 to 10 (10 being extremely frustrated), how infuriated do you become by waiting for the bus?
9. If you arrive late for your bus, what is the reason?
10. What service do you primarily use to find route information? If the answer was phone application, a new set of questions were asked in addition
 - (a) Which phone do you use?
 - (b) Which application do you primarily use?



- (c) On a scale from 1 to 10 (10 being super happy), how satisfied are you with the application of your choice
- (d) How did you hear about the application?
- (e) Please elaborate on your experiences with the application (Voluntary)

E.2 Waiting Time Survey Answers

This chapter contains the responses to the preliminary study. Responses are divided into three tables, and the reference to the participants makes it easy to cross reference one participant's answer throughout the tables. Table E.1 contains the answers regarding bus usage, Table E.2 is about waiting time, and Table E.3 contains responses about mobile application usage. Last, a list contains responses to the question *Describe your experience with the app*, which was an optional question.

Table E.1: Introductory and Bus Usage Questions

Answer nr	1	2	3	4	5
1	1-2	Student	18-20	Trondheim	To/from school, To/from the city, Visit
2	3-4	Student	21-23	Trondheim	To/from school
3	3-4	Student	21-23	Trondheim	To/from the city
4	1-2	Student	21-23	Trondheim	To/from the city, Airport express, Visit
5	1-2	Student	26-27	Trondheim	To/from school
6	3-4	Unemployed	21-23	Trondheim	To/from the city
7	7-9	Student	24-25	Trondheim	To/from school, To/from work, To/from the city



E.2. WAITING TIME SURVEY ANSWERS

8	1-2	Employed	21-23	Oslo	To/from work
9	1-2	Student	24-25	Trondheim	To/from the city, Airport express
10	1-2	Student	21-23	Oslo	To/from school, To/from work, Visit
11	1-2	Student	21-23	Oslo	To/from the city
12	3-4	Student	21-23	Trondheim	To/from school, To/from work, To/from the city
13	5-6	Employed	24-25	Oslo	To/from work, To/from the city
14	1-2	Student	21-23	Trondheim	To/from the city, Airport express, Visit
15	1-2	Student	18-20	oslo	To/from school
16	1-2	Employed	28-30	Oslo	To/from work
17	7-9	Employed	40<	Oslo	To/from work
18	1-2	Running business a	40<	Oslo	To/from the city
19	1-2	Student	24-25	Trondheim	To/from the city, Airport express
20	1-2	Student	26-27	Trondheim	To/from the city
21	10-12	Student	21-23	Tønsberg	To/from school, To/from the city
22	13-16	Student	21-23	Oslo	To/from school



APPENDIX E. PRELIMINARY STUDY

23	7-9	Employed	26-27	Oslo	To/from work, To/from the city, Airport express, Visit
24	10-12	Employed	40<	oslo	To/from work
25	7-9	Student	21-23	Trondheim	To/from school, To/from the city
26	3-4	Student	24-25	Trondheim	To/from work, To/from the city, Airport express
27	1-2	Student	21-23	Trondheim	Airport express
28	<1	Student	26-27	Oslo	To/from work
29	1-2	Student	24-25	trondheim	Visit
30	10-12	Employed	24-25	Oslo	To/from work
31	10-12	Employed	24-25	Oslo	To/from work, To/from the city, Visit
32	1-2	Student	21-23	Trondheim	To/from the city
33	10-12	Student	21-23	Trondheim	To/from school
34	<1	Student	21-23	Trondheim	To/from the city, Visit
35	17+	Employed	40<	oslo	To/from work
36	<1	Student	24-25	Trondheim	To/from the city
37	1-2	Student	21-23	Trondheim	To/from the city, Airport express
38	5-6	Employed	26-27	Trondheim	To/from work, Visit



E.2. WAITING TIME SURVEY ANSWERS

39	<1	Student	21-23	Oslo	Home to parents every other week
40	1-2	Student	21-23	Trondheim	Workout
41	<1	Employed	28-30	Trondheim	To/from the city
42	5-6	Student	21-23	Sarpsborg	To/from school, To/from the city
43	<1	Employed	30-34	Trondheim	To/from the city
44	13-16	Employed	26-27	Trondheim	To/from work
45	<1	Student	21-23	Trondheim	To/from the city
46	1-2	Employed	26-27	Trondheim	To/from the city
47	13-16	Student	26-27	Trondheim	To/from school
48	3-4	Employed	26-27	Trondheim	To/from work
49	<1	Employed	28-30	Gardermoen	Airport - Hotel
50	1-2	Student	21-23	Trondheim	Workout
51	<1	Employed	21-23	Trondheim	Visit
52	7-9	Student	21-23	Trondheim	To/from school
53	<1	Employed	21-23	Trondheim	Airport express
54	<1	Employed	40<	Molde	To/from work
55	<1	Employed	18-20	Trondheim	Airport express



Table E.2: Waiting on the Bus

Answer nr	6	7	8	10	9
1	5-6 minutes	Yes	6	App	Got wrong route information from the app
2	0-2 minutes	Yes	8	Route map at stop	The bus departs so often that I do not care if I'm one minute late
3	3-4 minutes	Yes	7	App	I am never too late
4	7-8 minutes	Yes	9	App	Miscalculation of walking time to the bus stop
5	5-6 minutes	Yes	8	App	Real-time was inaccurate and showed the wrong time
6	5-6 minutes	Yes	3	Route map at stop	Did not use the route information
7	5-6 minutes	Do not know	5	App	Miscalculation of walking time to the bus stop
8	3-4 minutes	No	2	Information screen at stop	The bus departs so often that I do not care if I'm one minute late
9	9-10 minutes	Yes	3	Information screen at stop	Did not use the route information
10	5-6 minutes	Yes	10	App	Real-time was inaccurate and showed the wrong time



E.2. WAITING TIME SURVEY ANSWERS

11	3-4 minutes	Yes	4	App	Real-time was inaccurate and showed the wrong time
12	3-4 minutes	Yes	5	App	Miscalculation of walking time to the bus stop
13	3-4 minutes	Yes	5	App	Real-time was inaccurate and showed the wrong time
14	7-8 minutes	Yes	4	App	Got wrong route information from the app
15	3-4 minutes	Yes	2	Web page	Real-time was inaccurate and showed the wrong time
16	3-4 minutes	Yes	2	App	Miscalculation of walking time to the bus stop
17	5-6 minutes	No	2	App	Miscalculation of walking time to the bus stop
18	9-10 minutes	Yes	8	App	Real-time was inaccurate and showed the wrong time
19	3-4 minutes	Yes	2	Web page	The bus departs so often that I do not care if I'm one minute late
20	9-10 minutes	Yes	8	Web page	Real-time was inaccurate and showed the wrong time



APPENDIX E. PRELIMINARY STUDY

21	7-8 minutes	Do not know	5	App	Did not use the route information
22	3-4 minutes	Yes	3	Web page	Did not use the route information
23	3-4 minutes	Yes	8	Web page	Miscalculation of walking time to the bus stop
24	0-2 minutes	Yes	5	App	The bus departs so often that I do not care if I'm one minute late
25	3-4 minutes	Yes	5	App	Got wrong route information from the app
26	7-8 minutes	Yes	8	App	Got wrong route information from the app
27	7-8 minutes	Yes	7	App	The bus was ahead of schedule
28	0-2 minutes	No	3	Web page	The bus departs so often that I do not care if I'm one minute late
29	5-6 minutes	Yes	7	Web page	Miscalculation of walking time to the bus stop
30	3-4 minutes	Yes	5	App	Miscalculation of walking time to the bus stop
31	3-4 minutes	Yes	4	App	Miscalculation of walking time to the bus stop



E.2. WAITING TIME SURVEY ANSWERS

32	5-6 minutes	Yes	7	App	Real-time was inaccurate and showed the wrong time
33	0-2 minutes	No	8	App	Real-time was inaccurate and showed the wrong time
34	9-10 minutes	Yes	9	Web page	Got wrong route information from the app
35	7-8 minutes	Yes	10	Timetable booklet	Miscalculation of walking time to the bus stop
36	3-4 minutes	Yes	6	App	I was slow out the door
37	5-6 minutes	Yes	4	App	Miscalculation of walking time to the bus stop
38	5-6 minutes	Yes	6	Information screen at stop	Miscalculation of walking time to the bus stop
39	11-14 minutes	No	9	Web page	Miscalculation of walking time to the bus stop
40	3-4 minutes	Do not know	6	Web page	The bus was ahead of schedule
41	11-14 minutes	Yes	5	Web page	I am never too late
42	9-10 minutes	Yes	10	Web page	The bus was ahead of schedule
43	5-6 minutes	Yes	8	Web page	Miscalculation of walking time to the bus stop
44	3-4 minutes	Yes	5	Information screen at stop	Miscalculation of walking time to the bus stop



APPENDIX E. PRELIMINARY STUDY

45	3-4 minutes	Do not know	2	Web page	Did not use the route information
46	5-6 minutes	Yes	7	Timetable booklet	Miscalculation of walking time to the bus stop
47	3-4 minutes	Yes	7	Information screen at stop	Real-time was inaccurate and showed the wrong time
48	9-10 minutes	Yes	10	App	I am never too late
49	5-6 minutes	Yes	5	Information screen at stop	Did not use the route information
50	5-6 minutes	No	3	App	The bus departs so often that I do not care if I'm one minute late
51	5-6 minutes	Yes	2	Web page	Miscalculation of walking time to the bus stop
52	3-4 minutes	Yes	8	Timetable booklet	Miscalculation of walking time to the bus stop
53	0-2 minutes	Yes	5	Web page	Miscalculation of walking time to the bus stop
54	3-4 minutes	Yes	5	Web page	Miscalculation of walking time to the bus stop
55	Do not know	Yes	10	App	The bus was ahead of schedule



Table E.3: App Questions

Answer nr	10a	10b	10c	10d
1	iPhone	Bartebuss	8	Friends
3	iPhone	Bartebuss	5	Friends
5	iPhone	Bartebuss	6	Friends
7	iPhone	Bartebuss	6	Friends
10	Android	RuterReise	7	Do not know
11	iPhone	Bartebuss	3	Friends
12	Android	Bartebuss	5	Friends
13	iPhone	RuterReise	5	Appstore search
14	Android	RuterReise	5	Friends
16	Windows phone	Trine i farta	8	Friends
17	Android	RuterReise	8	Friends
18	Android	RuterReise	5	Web search
21	Android	VKT	6	Web search
24	Android	RuterReise	9	Appstore search
25	Android	AtB sanntid	5	Add
26	Android	Bartebuss	8	Friends
27	iPhone	Bartebuss	6	Friends
30	Android	RuterReise	3	Appstore search
31	iPhone	RuterReise	6	Add
32	iPhone	Bartebuss	8	Friends
33	Android	Bartebuss	8	Friends
36	Android	AtB reiseplanlegger	6	Appstore search
37	Android	AtB reiseplanlegger	5	?
48	Android	AtB sanntid	6	Friends
50	Android	Bartebuss	5	Friends
55	iPhone	Bartebuss	4	Friends

This list contains answers to the question *Describe your experience with the app*, sorted by the participant number.

- **#10 RuterReise** It needs a lot more features, such as being able to see the entire bus route (with stops) when you press a route.
- **#17 RuterReise** When you scale from 1 to 10, it should say what is worst, best, etc.. Such as the previous question. Eg. the one about frustration could easily be clarified.



- **#24 RuterReise** Very good! Definitely a "must have" app.
- **#26 Bartebuss** Very good experience with the app, particularly fond of the UI and the way information is displayed. Could have been better at updating time when it comes to major delays, but expect that some of this lies with AtB's real tables ...
- **#30 RuterReise** The way developers think I use the app is quite banal. For example cumbersome to find the bus from A to B, but easy to find when the next bus passes. When the next bus passes I do not care if I have to wait 20 minutes at the bus exchange.
- **#33 Bartebuss** The app frequently change the time the bus will arrive when it is approaching, so it comes sooner than you think. This makes it difficult to calculate when to go home.
- **#55 Bartebuss** Cumbersome to identify the timetable for a single bus, should have been a function to select the bus you want and not just a menu for bus stops you want.

