# MultiBRIS: A Multiple Platform Approach to the Ultimate Bus Route Information System for Mobile Devices

Runar Andersstuen, Trond Bøe Engell

December 21, 2011

**Abstract**

We describe MultiBRIS, a multipleplatform approach to the ultimate *bus route information system* for mobile devices. The system is *context aware*, which means that users only need to tell the system where they wish to go, and the system takes care of the rest. The user is presented with a list of possible routes he or she can take to reach the desired destination. The results are also shown on a map that makes finding the bus-stops very easy. This functionality is made available through an application that can be run on *multiple platforms*, with a *minimal amount of data transfer and calculation needed* on the client side.

A state-of-the-art survey for existing public transport information systems was performed. Based on the results, we decided how to best make use of the existing technology in order to create the MultiBRIS system described here.

The prototype system that was created consists of two parts, the MultiBRIS server and the MultiBRIS client. The client has a minimal amount of business logic implemented. It focuses instead on the interaction with the user and facilitates multiple platform possibilities , using technology like *HTML5, PhoneGap and Sencha Touch*. The MultiBRIS server handles most of the business logic and communicates with external services.

Finally, we present ideas on how to future develop and expand the system so it can reach beyond Trondheim and incorporate more functionality with help from the field of artificial intelligence.

# Contents

# 1 Introduction

The project was given by Tore Amble and IDI. The grand goal as presented by Tore Amble is to make an ultimate bus route information system for the future. Below is the original assignment text as it was presented to this project:

> *FUIROUS - Fremtidens ultimate intelligente ruteopplysningssystem.*
>
> *BusTUC is a natural language bus route system for Trondheim. It gives information about scheduled bus route passings, but has no information about the real passing times. This is about to change, because Team trafikk has installed GPS tracking of the buses, giving access to real passing times and delays. Besides, with new smart phones arriving rapidly on the market, there are possibilities for GPS localisations and connections to maps. The project shall take a broad view, and consider all possible advanced concepts, resulting in advanced smart phone applications.*

Part of the goal and the key focus of this report, is researching existing applications and to develop a prototype multi-platform bus route information application for mobile devices. The project is based on an master thesis called "An intelligent smartphone application: Combining real-time with static data in pursuit of the quickest way to travel by bus" by Magnus Raaum [13]. The work done in his thesis is based around an application specifically made for a single platform called Android[1]. The basic idea behind this project is the prediction that in order to get closer to an ultimate bus route information system, the system has to be able to run on multiple platforms to be available to as many users as possible. If the system is easily adaptable to new platforms, it is much easier to reach this goal. In order to realise this, it is essential that the system relies on technology standards that are widely accepted and implemented. Our prototype system is called MultiBRIS.

## 1.1 Terminology and Abbreviations

This section describes the terminology and abbreviations used in this paper. Some of the explanations below are partially taken from sources such as Wikipedia[2], product web-sites, and standardisation organisations web-sites.

[1] `http://www.android.com/`

[2] `http://en.wikipedia.org/wiki/Main_Page`

- **API** - An Application Programming interface is a source code based specification intended to be used as an interface by software components to communicate with each other.
- **Apple WebKit** - A layout engine designed to allow web browsers to render web pages. It is used in the default browser on both Android and iOS
- **AtB** - Administration agency for public transport in Sør-Trøndelag. Formerly Team Trafikk.
- **Black Box** - Black box is an object which can be viewed solely in terms of its input, output and transfer characteristics without any knowledge of its internal workings.
- **Business Logic** - Business logic is a non-technical term generally used to describe the functional algorithms that handle information exchange between a database and a user interface. In this paper, it is specifically used to describe the part of the system that makes the actual computations and calls external services.
- **BusTUC** - BusTUC is a natural language problem solver capable of answering questions about bus departures in Trondheim stated in common English.
- **CEO** - Chief Executive Officer. He/she is the highest-ranking corporate officer (executive) or administrator in charge of total management of an organization.
- **CPU** - Central Processing Unit is the portion of a computer system that carries out the instructions of a computer program, to perform the basic arithmetical, logical, and input/output operations of the system.
- **DOM**- Document Object Model is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents.
- **GPS** - Global Positioning System is a orbit-based satellite navigation system that provides location and time information in all weather, anywhere on or near the Earth, where there is an unobstructed line of sight to four or more GPS satellites.
- **GPU**- Graphical Processing Unit is a specialized circuit designed to rapidly manipulate and alter memory in a way whichs accelerate the building of images in a frame buffer intended for output to a display.

- **GUI** - Graphical User Interface is a type of electronic user interface that allows users to interact with images rather than text commands.
- **HTML** - HyperText Markup Language is a markup language for formatting web pages.
- **HTTP** - HyperText Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems.
- **HTTP GET** - Part of the HTTP, requests a representation of the specified resources.
- **IDI** - Department of Computer and Information Science (Institutt for datateknikk og informasjonsvitenskap), NTNU.
- **IP**- Internet Protocol is the principal communications protocol used for relaying packets across an internetwork. Responsible for routing packets across network boundaries, it is the primary protocol that establishes the Internet.
- **JSON** - JavaScript Object Notation is a lightweight data-interchange format.
- **MultiBRIS** - "Multiple-platform approach to the Ultimate Bus Route Information System" is a system developed in parallel to TABuss.
- **NTNU** - Norwegian University of Science and Technology.
- **OS** - Operating System is a set of programs that manage computer hardware resources and provide common services for application software.
- **PHP** - PHP is a general-purpose server-side scripting language.
- **SMS** - Short Message Service is a text messaging service component of phone, web, or mobile communication systems.
- **SOAP** - Simple Object Access Protocol is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP)for message negotiation and transmission.
- **TABuss** - "Tore Amble Buss" is an intelligent Android bus route application.
- **TUC** - The Understanding Computer is a reasoning system developed at IDI by Tore Amble.

- **UI** - User Interface, facilitates interaction between humans and machines.
- **URL** - Uniform Resource Locator is a specific character string that constitutes a reference to an Internet resource.
- **WireShark** - Wireshark is a network protocol analyser. It lets the user capture and interactively browse the traffic running on a computer network.
- **XML** - Extensible Markup Language is a markup language for sharing structured data.

# 2 Goals and Methods

The goal is to make a multi-platform prototype system. Through the development of this system it is also a goal to reveal strengths and weaknesses around the "web-application approach" to making an application for mobile devices. The main goals are given below to describe the bounds of the project.

## 2.1 State-of-the-Art Survey

The goal for this state-of-the-art survey is to give an overview over the latest bus route systems and the available technology for mobile device software development. A set of bus route systems that exist for mobile devices, and the functionality that these systems implement, are reviewed to see what technology has been applied and what these technologies can contribute in this project with respect to multi-platform systems. The information gathered from the survey reveals valuable information that is to be used later in the project.

### 2.1.1 Technologies and Design

The goal is to collect information on what technologies exist for creating multi-platform applications on mobile devices. A typical technology related challenge in multi-platform applications is how to make use of device-native sensors such as GPS. Other typical areas of importance when working on mobile devices are the assessment of how much data should be *transferred* in device communication and how much of the business logic should reside on the clients. This survey makes a conclusion based on whether the existing technology is sufficiently mature for use in our problem domain.

### 2.1.2 Existing Applications

The goal is to review a set of diverse applications both in the Trondheim area and in the rest of the world that promise to aid people that travel by bus. Each application is reviewed in order to get an overview of what technology is being used today to create these and how functionality has been implemented accordingly.

## 2.2 Prototype System

The best way to prove that something works is by creating a working sample. Hence, the goal is to make a prototype system that has the same functionality as the application developed for the Android platform in the master thesis by Magnus Raaum[13]. The main functionality of the android application is that it automatically finds out where the traveler is located,

so that the traveler only has to input the desired destination information in order to get the bus route suggestions. It also displays a geographical map, where the closest bus-stops are shown. For more information on the specifications the reader is referred to Raaum's master thesis[13].

As the source code from the thesis was available for this project, the main challenge was that the prototype should be a web-application and not device-native throughout. This means that new technology is be used for everything regarding the graphical user-interface. The business-logic that is implemented in the existing Android application could be used to some extent, but some design choices was to be made concerning what to implement on the server-side and what to implement on the client-side.

## 2.3 Collaboration with Project TABuss

The FUIROUS project actually consists of two groups, the other project group was given the working name *TABuss*[9]. Both groups had Magnus Raums master thesis (2011) as a starting point[13]. TABuss was working on improving the Android application that Magnus developed in his master thesis, while we were looking at the possibilities for a multi platform version of the application. As both groups had the same code as a starting point, we shared information about known bugs in the business logic. In the later phase of the project, we developed a server to handle much of the business logic that was earlier on the client side. As can be seen from the result section of this report, reducing query time and the need for data transfer was imperative to make a client-application viable. Therefore the TABuss project started using our server, and by doing so giving us valuable feedback on bugs and ideas on improvement.

# 3 State of the Art Survey

This section describes the current state-of-the-art technologies for mobile device software development and review existing systems based on these technologies.

## 3.1 Technology

This section takes a look at multi-platform strategies for software development on mobile devices and decides which one is best suited for this project.

### 3.1.1 The Age of Smartphones

The worldwide smartphone[3] market has expanded immensely during the last few years. There were 440 million mobile devices sold by vendors in the 3rd quarter of 2011[4]. Of these, 115 million were smartphones. This equals a marked share of 26.1%. The marked share has increased continuously the last few years, as depicted in the diagram below. Companies like Apple, HTC and Samsung, to name a few that have focused on developing smartphones, have gained large parts of the market share. Other companies, like Nokia, that are big on mobile phones, seem to be on a negative trend.

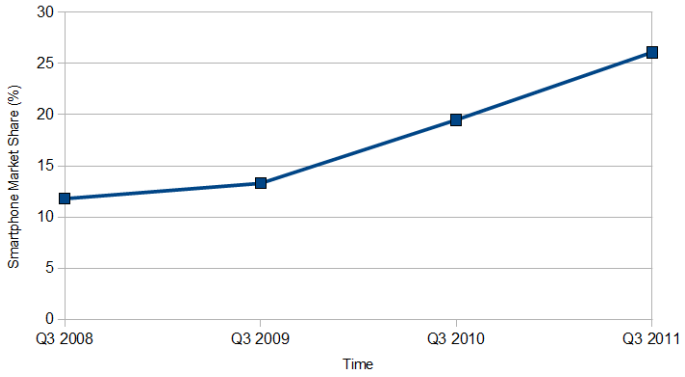


Figure 1: Percentage of worldwide mobile device sales to end users that are smartphone devices. Source: Gartner.

[3] `http://en.wikipedia.org/wiki/Smartphone`

[4] Deducted by the numbers given in Gartner's press releases :
Q3 2011: `http://www.gartner.com/it/page.jsp?id=1848514`
Q3 2010: `http://www.gartner.com/it/page.jsp?id=1466313`
Q3 2009: `http://www.gartner.com/it/page.jsp?id=1224645`

Other big trends in the mobile market is the rapid growth of the mobile internet connectivity and social networks(2010)[8]. Internet Data Corporation predicts that by 2015, more Internet users in United States will access Internet through mobile devices than through PCs or other wireline devices[5].

As smartphones take over the marked, the need for mobile-enabled content and services increases. Unfortunately, huge amounts of available devices has split the market into several mobile technology platforms. Leading platforms like iOS[6], Android[7] and RIM[8] are all based on different operating systems and code languages. Software developers need to make choices for which platform to support and then learn the native language of that platform. If they want to focus on several platforms and reach out to a larger audience, duplicate efforts are needed to implement specific software on each platform and keep maintaining each code base separately. Consequently, application development time can be immense.

One of the reasons why smartphones are popular, is that people always bring their phones with them, wherever they go. The mobility of smartphones opens up for new uses that stationary PCs cannot. People are getting used to doing most things without the constraints of place. An example of this is reading books and journals. Many would prefer downloading texts directly to their mobile device from the comforts of their home rather than hauling around stacks of physical books and magazines from libraries. In the field of scholar communication, this is a problem that needs to be addressed [11]. In this paper, Richard Padley describes the challenge of bringing publishing of scholarly work to a range of mobile devices. As publishers convert books and journals to digital form[9] for easier access, new challenges appear: How to reach the masses? Commercial uncertainties in the highly competitive mobile market space cause questions to appear when approaching the challenges of cross-platform publishing. How are they supposed to know which platforms will succeed? What resources and knowledge are needed to make a sensible decision? A lot of factors need to be taken into account, like Apple losing Steve Jobs as CEO, Google's future vision for Android and so on. Such organisation-wide strategic decisions can not to be taken lightly.

Fortunately, there is a way around this issue: Multi-platform development. The big advantage here is a single code base, which reduces application development time greatly. There are several ways of developing multi-platform software. Determining which strategy is most suitable can be a challenge. What is certain though, is that project requirements must

[5]IDC Press Release : `http://www.idc.com/getdoc.jsp?containerId=prUS23028711`

[6]`http://www.apple.com/ios/`

[7]`http://www.android.com/`

[8]`http://www.rim.com/`

[9]`http://en.wikipedia.org/wiki/E-book`

the primary deciding factor. The following paragraphs takes a look at a few strategies to achieve multi-platform applications:

**Web Applications** Web-based applications make use of HTML5[10], CSS[11] and Javascript[12] to create mobile websites that aim to look and feel like a native mobile application. Web applications can use JavaScript frameworks such as Sencha Touch[13] and JQuery Mobile[14], that are solely designed for mobile development , to replicate mobile user interfaces. Web applications can be conveniently run in a web browser[15] and is therefore essentially multi-platform since most mobile device are equipped with web browsers these days. The disadvantages of web applications are the limited access to device-specific features, like sensors, and product publishing. They can not be uploaded to application stores, like Android Market[16] and iOS App Store[17], which could have a negative effect on availability and product sales.

**Proprietary Middleware** Applications can also be based on web services like Red Foundry[18]. Developers gets access to a web interface where an application is graphically created by selecting a set of prebuilt modules. When the developer has picked all the modules that provide the necessary functionality, the service builds a native application that the developer can submit to an application store or market. The advantage of this strategy is that the developer do not need any specialist knowledge or programming experience to create applications that look good and perform well. The drawback is that the services usually are not free and that the design and functionality are limited to what is offered in the service.

**Native Applications and Hybrid Applications** An application is referred to as *native* if it is written in a specific code language and if it designed to run in a specific operating system. The main advantage of the native applications is that they work as intended by the operating system developers. Device features like sensors, contact list and storage are easily accessed directly and the llibraries offered by the application programming interface (API)[19] are optimised for the specific operating system.

[10] `http://en.wikipedia.org/wiki/HTML5`
[11] `http://no.wikipedia.org/wiki/Cascading_Style_Sheets`
[12] `http://en.wikipedia.org/wiki/JavaScript`
[13] `http://www.sencha.com/products/touch`
[14] `http://jquerymobile.com/`
[15] `http://en.wikipedia.org/wiki/Web_browser`
[16] `http://en.wikipedia.org/wiki/Android_market`
[17] `http://en.wikipedia.org/wiki/App_Store_(iOS)`
[18] `http://www.redfoundry.com/`
[19] `http://en.wikipedia.org/wiki/Application_programming_interface`

The Hybrid applications are written as web applications, using coding technologies such as HTML5, CSS and JavaScript. The web applications are then wrapped by one of the available "multiple phone web-based application frameworks"[20] such as PhoneGap in order to emulate native behaviour. Device features like sensors, contact list and storage are provided by these platforms. Unlike the other alternatives which are confined to browsers and have limited functionality, the hybrid strategy takes, as Pradley puts it (2011), *"the best of two worlds"*, referring to native- and web applications[11]. Developers get a greater control over application design. They use one single code base, but still get access to device features. Christ (2011) states *"The hybrid approach, which even in its infancy, is a strong solution. When developers use the hybrid approach thoughtfully, the result can be comparable to a native application-a true bridging of the gap from native to web-based applications"*[2]. For a more in-depth comparison between native and hybrid applications the reader is referred to section 2.2 of the other FUIROUS project report TABuss[9]. Multiple phone web-based application frameworks are from now on referred to as *Deployment technologies* in this paper.

**Big Influencial Companies go for HTML5** Adobe Flash[21] is a proprietary multimedia platform used to add animation, video, and interactivity to web pages. It has also become a tool for creating web applications that mimic many characteristics of desktop applications. The last few years, Adobe has also focused on adding support for their Adobe Flash Player[22] to mobile devices. Support has been added for a few devices, but Adobes future plans for the Adobe Flash Player on mobile devices has come to a halt. Apple's[23] unwillingness to add support for Adobe Flash Player to their products marks the start of Adobe Flash Player's downfall for mobile devices. Steve Jobs stated in a 2010 press release[24]: *"The mobile era is about low power devices, touch interfaces and open web standards - all areas where Flash falls short."* He also states that Apple will rather focus on supporting open platforms like HTML5. Since then Adobe has stopped further development of the Adobe Flash Player for mobile devices and instead committed itself to HTML5, a platform with broader support and capabilities than Flash was ever able to deliver. This is reflected by their recent (2011) acquisition of Nitobi[25], which spesialise in the deployment technology: PhoneGap[26].

[20] http://en.wikipedia.org/wiki/Multiple_phone_web-based_application_framework

[21] http://en.wikipedia.org/wiki/Adobe_Flash

[22] http://www.adobe.com/no/products/flashplayer.html

[23] http://www.apple.com

[24] http://www.apple.com/hotnews/thoughts-on-flash/

[25] http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html

[26] http://phonegap.com/

## 3.2 Existing Applications in Trondheim

The Android and iOS systems were selected as our test platforms because the phones available for testing were based on these. These are popular in Norway, so it was expected to find a broad range of existing applications. Research was done on the web, Android Market and Apple App Store to find suitable participants for this application comparison. The applications where chosen in such a way that both Android and iPhone platforms are represented, ranging from simplistic applications to more sophisticated ones with lots of functionality. The reviews are by no means thorough, and were mainly written in order to map the functionality of each application to get an overview of what technology and functions are needed in a smartphone bus route application by present standards. Note that these applications were updated to the newest versions by the 5th of October 2011. Changes in the applications after this date are not taken into consideration.

This survey was done in the Software Lab on the campus of NTNU, using the smartphones Samsung Galaxy S2 and iPhone 3GS.

### 3.2.1 "Bussorakel"

"Bussorakel" is a native Android application created by Erlend Klakegg Bergheim and uses the BussBuddy API[27]. It is a very simple application with minimalistic design and functionality.

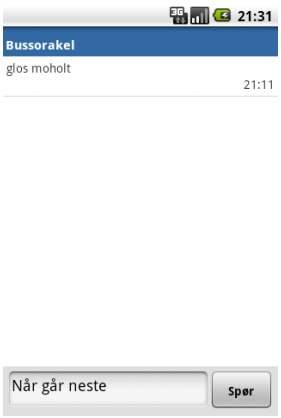

Figure 2: Bussorakel.

Although simple design may open up for efficient use, it does seem a bit crude and did not give us any incentives to choose to use this particular application. It consists merely of a text field for sending bus route queries to the Bus Oracle"[28] and that was all. It is easy to use, but the response time is slow and would certainly become a stress factor for any person in a hurry. There is no *favourite* functionality, but the system logs the user's queries until manual deletion. The application gives the user small helpful features through the menu, though. For instance, when the user receive the answer from the Oracle it can be posted on any installed social media, e-mail or SMS.

[27] `http://api.busbuddy.no/`
[28] `www.atb.no`

### 3.2.2 Bartebuss

Bartebuss[29] is created in HTML5 by Rune M. Andersen and uses the BussBuddy API. It is very rich in functionality, consisting of favourites, a view of nearby bus-stops, alphabetically ordered group search of bus-stops, oracle queries and a map view. The application consists of five ways to find the bus times. The *favourites* tab, where several bus stops can be saved, gives quick access to the routes often used. When a *favourite* bus-stop is selected from this list, real-time data for this bus-stop is displayed. The second way to use the application is an option called "Near me", where the closest bus-stops are listed and can be selected the same way as in favourites. The third option is "Search" where the user can manually search for a specific bus-stop alphabetically, and show its real-time data. The fourth option is the use of the Bus Oracle, where queries like "When does the bus go from Gløs to Munkegata" give the user several route suggestions. The last option is to select a bus-stop on the map, which shows real-time data of all busses going through that stop. The map view does not pinpoint the user's location and it is a bit slow. Whether this is because the map technology is OpenStreetMap[30] or because the application is written in HTML5 code is still not clear and left as future work to explore. The user interface is created in a orderly and intuitive way. The colour usage is easy on the eyes. The diverse functionality makes it perfect for the advanced bus traveler, but for the casual user the functionality may be too overwhelming.


Figure 3: Bartebuss.

### 3.2.3 Busstider

Busstider[31] is a native Android application made by Martin M. Syversen and is developed using the BussBuddy API. It consists of the Bus Oracle functionality and a Google Maps implementation.

This application is in the mid-range in our comparison. It offers a pleasant amount of functionality together with a clean design. The user can either send queries to the Bus Oracle or the map can be used, where the user can see the current position and pick the start and end destina-


Figure 4: Busstider.

[29] http://www.bartebuss.no
[30] http://www.openstreetmap.org/
[31] http://www.a2bsoft.net/

tion. The response time is good. There is no *favourite* functionality, but the system logs the user's queries until they manually deleted.

### 3.2.4 Alf's ByBuss

Alf's ByBuss[32] is a native Android application which uses the BusBuddy API. It is made by Alf Simen Sørensen and consists of a Google Map view that sends queries to the Bus Oracle. The application has a map and a text-box for entering queries to the Bus Oracle. The map loads all the bus-stops in Trondheim and finds the user's position. The user may then select the start and end bus-stop in order to get the bus route suggestions. An additional feature when the user selects a bus-stop is to view the real-time data of buses going in either direction through that bus-stop. The "Use your address" menu option will find the closest bus-stop and put it in the textbox as the departure stop. This reduces the amount of user-interaction needed before posting a query. Other functions include a "reverse route" and a "download the AtB route brochure" feature. This application is perhaps the most intuitive and responsive of all the reviewed applications.

Figure 5: Alf's ByBuss.

### 3.2.5 Bussdroid

Bussdroid[33] is a native Android application created by Ken Børge Viktil. It has real-time data for bus-stops, an oracle query feature and the ability to store queries as favourites.

In order to use the application the user can make use of the "Real-time" view and make a search for a bus-stop. The user will then be presented with real-time data for buses going through the bus-stop. The user can also simply ask the Bus Oracle as in many of the other applications. The user can then store this query as a *favourite*. The lack of a map may be a deal-breaker for people who are not familiar with Trondheim. It is very responsive and uses swipe technology to switch between the functionality views. Swiping is fun and intuitive.

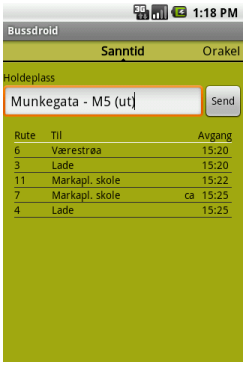

Figure 6: Bussdroid.

[32] http://bybuss.alfsimen.com/

[33] https://market.android.com/developer?pub=Ken+B%C3%B8rge+Viktil

### 3.2.6 BusApp Trondheim

BusApp Trondheim[34] is a native Android application created by Skogvold Android that focuses on finding bus-stops and real time departure times in Trondheim.

Not much effort is put into this application's GUI. It is presented in such a way that it is looks like the author just threw a few textboxes and buttons together. Lack of Oracle support also makes it much less user friendly than the other applications. This application does not have route suggestions at all. It merely functions as a bus-stop finder. It also has the real-time feature, but it uses SMS to communicate with AtB, which is expensive and a bit awkward. This is an outdated solution.


Figure 7: BusApp Trondheim.

### 3.2.7 Bussruter

Bussruter[35] is a native Android application written by Stian Standahl. It is by far the simplest of all the applications in this comparison. The only function it provides is to list the available buss route brochures from AtB's webpages and let the user download them.

The application has two views. First the user goes to the "All bus routes" view where the available brochures get listed. When the user clicks on one, it gets downloaded. Then the user can view the downloaded brochures in the "Downloaded brochures" view.


Figure 8: Bussruter.

[34] https://market.android.com/developer?pub=Skogvold+Android
[35] https://market.android.com/details?id=com.bussruter&hl=no

### 3.2.8 Bussøye

Bussøye is a native iPhone application developed by Capgemini Norge AS[36]. It uses real-time data to give information about when the next bus arrives at any selected bus-stops in Trondheim.

This application lets the user search for bus-stops and displays where they are located on a map. It also have the ability to only show the nearby bus-stops. The *favourites* function works well and the user's search history is also stored. The application has a GUI that is easy to use, responsive and works well. The bus route requests are also responsive. The only drawback of this application is the lack of Oracle support and lack of real-time support. Otherwise it works very well.


Figure 9: Bussøye.

### 3.2.9 Comparison Charts

| | Buss-Orakel | Barte-Buss | Buss-tider | Alf's By-buss Trond-heim | Buss-droid | Bus-App Trond-heim | Buss-ruter | Buss-øye |
|---|---|---|---|---|---|---|---|---|
| Plattform | A/iP | HTML5 | A/iP | A | A | A | A | iP |
| Multi-Language | Yes | No | No | Yes | No | Yes | No | No |
| Cost | Free | Free | Free | Free | Free | Yes | No | Free |
| Favourites | No | Yes | No | No | Yes | No | No | Yes |
| History | Yes | Yes | Yes | Yes | No | Yes | No | Yes |
| Route PDF down-load | No | No | No | Yes | No | No | Yes | No |
| Map | - | OSP | GM | GM | - | GM | - | GM |
| Shows closest bus-stops | - | Yes | No | No | No | Yes | No | Yes |
| Uses GPS | - | Yes | Yes | Yes | No | Yes | No | Yes |

Table 1: List of attributes, GM: GoogleMap, OSP: Open Street Map, A: Android, iP: iPhone.

[36] `http://www.no.capgemini.com/`

## 3.3 Existing Applications in Other Parts of the World

This subsection describes foreign systems that are relevant to this project.

### 3.3.1 Google Transit

The Google Transit system essentially consists of two parts. One first part relates to the consumer of the service and is called *Google Transit Trip Planner* . The other part is called *General Transit Feed Specification*[37]. It is used by data providers to feed the Google Transit Service with data. Users of GTFS are typically public transportation agencies.

The Google Transit Trip Planner lets users pose queries to the transit system in three different forms. By address, by location name and by directional indicators (i.e. NE, NW, SE, SW) or by GPS coordinates. All of these query types are accompanied by date and time[38]. It is also possible to combine these different forms of querying. A result from Google Transit Trip Planner displays both text and directional lines on Google Maps[39]. Figure 10 shows the result of a combined query to Google Transit Trip Planner containing stop name and GPS coordinates.

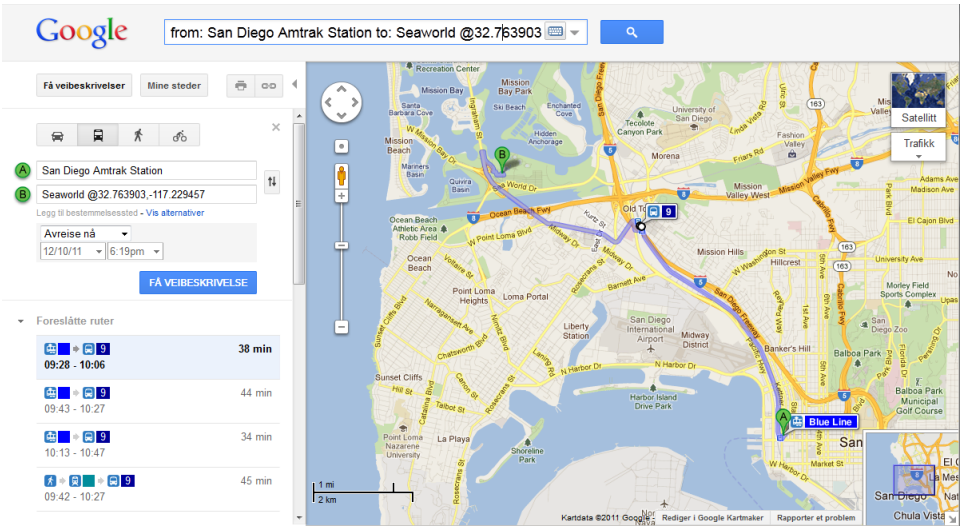


Figure 10: Google Transit Planner Result.

GTFS General Transit Feed Specification defines a common format for public transportation schedules and associated geographic information. The

[37] `http://code.google.com/intl/no/transit/spec/transit_feed_specification.html`

[38] `https://spreadsheets.google.com/pub?key=puMHBiWYEbXT0UxQGLDpuBA&gid=11`

[39] `http://maps.google.com/`

data is fed to Google with a ZIP-file. This ZIP-file contains a number of txt-files. These txt-files contain comma separated values with information. Table 2 describes the txt-files and their content. As one can see there is substantial amount of information needed in order to start using this service from a transit agencies point of view. The biggest challenge for the agencies, provided that they actually have the information needed, is the transformation of data so that it fits the Google Transit Feed Specification[10].

| **File Name** | **Content** |
|---|---|
| agency.txt (required) | Contains information about one or more transit agencies that provide the data in this feed. |
| stops.txt (required) | Contains information about individual locations where vehicles pick up or drop off passengers. |
| routes.txt (required) | Contains information about a transit organization's routes. A route is a group of trips that are displayed to riders as a single service. |
| trips.txt (required) | Lists all trips and their routes. A trip is a sequence of two or more stops that occurs at specific time. |
| stop_times.txt (required) | Lists the times that a vehicle arrives at and departs from individual stops for each trip. |
| calendar.txt (required) | Defines dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available. |
| calendar_dates.txt (optional) | Lists exceptions for the service IDs defined in the calendar.txt file. If calendar_dates.txt includes ALL dates of service, this file may be specified instead of calendar.txt. |
| fair_attributes.txt (optional) | Defines fare information for a transit organization's routes. |
| fair_rules.txt (optional) | Defines the rules for applying fare information for a transit organization's routes. |
| shapes.txt (optional) | Defines the rules for drawing lines on a map to represent a transit organization's routes. |
| frequencies.txt (optional) | Defines the headway (time between trips) for routes with variable frequency of service. |
| transfers.txt (optional) | Defines the rules for making connections at transfer points between routes. |

Table 2: GTFS Files and their Content.

### 3.3.2 OneBusAway

OneBusAway [40] is a set of transit traveler information tools developed for providing real-time arrival information, a trip planner, a schedule and route browser, and a transit-friendly destination finder for Seattle area bus riders. Ferris, Watkins and Borning (2010) concentrated on the tools for providing real-time arrival information and extended the functionality to add support for location sensing. They first created an iPhone application to exploit its localization framework and built-in multitouch map support, but later also developed an experimental multi-platform web application for real-time arrival information based on JavaScript to meet user demands[4].

## 3.4 Conclusion

The FUIROUS projects have a big common vision: Becoming the ultimate bus route information system for the future.

The *ultimate* part of this vision tells us that it expects to become not only the best bus route information system out there, but also that it wants to have as many users as possible. The importance of a multi-platform application has been explained earlier. The languages HTML5, CSS and JavaScript achieves this for us. In order to reach out to users, though, the application also needs to be eligible for deployment into the variety of application stores out there, which provides opportunities for publishing, advertisement and collecting fees. Hybrid applications are designed for this, using deployment technologies. The disadvantages of the hybrid-solutions will dissipate as the deployment technologies, along with the browsers on the devices, mature and become more robust.

As for the *future* part of the vision: This is a research project. Software created here are prototypes. Therefore it is important, for further development of this prototype, that they are created within standards that are future proof. Big software companies like Apple and Adobe both see the potential of the open standards HTML5, JavaScript and CSS. These technologies are standards that has been around for many years, and are firmly set in the world wide web. The Web Hypertext Application Technology Working Group (WHATWG) states that preserving backwards compatibility with browsers designed for earlier versions of HTML is one of the key features of HTML5[41]. This feature has served them well this far, and will most likely do so for many years ahead.

The main function of the application review was to create a base of information for currently existing systems for bus route information. Inspiration was drawn from the applications to bring together all the strengths

[40] `http://onebusaway.org`

[41] `http://wiki.whatwg.org/wiki/FAQ`

and avoid the weaknesses when implementing our own prototype. Reinventing the wheel is a waste of time. It is crucial to not make the same mistakes as other developers in order for this project to be considered successful. As mentioned earlier, the primary goal in this project is to create a prototype that consists of all the functionality in Magnus Raaum's Android application[13]. The secondary goal is to take the next step in creating the ultimate[1] application. This step consists of implementing an appealing and intuitive graphical user interface. Good applications like "Bartebuss" and "Alf's Bybuss Trondheim" and small, helpful little features in the other programs are welcomed inspirational sources. It is important not to take it too far though, so that the resulting program get too complex and untidy. This could easily kill intuitivity and responsiveness, which are key properties of a successful application. An interesting thing to make note of is that the application that have been created by means of HTML5 and JavaScript, namely Bartebuss, do not hold back against the native applications. On the contrary, it is perhaps the most feature rich and well-working application of them all. This, of course, gives more incentives to make use of such technology.

The OneBusAway article provides valuable user feedback on an application much like the one Magnus Raums has made. The most interesting user comment is that the users want the system to provide a bookmark functionality. They want to be able to tap the bookmarked destination at any time and receive route suggestions to that bookmark from their current location[4]. Valuable feedback like this plays an important role when choosing functionality for our prototypes.

Google Transit offers a good way for uses to interact with their service directly. The drawback is the lack of APIs for external developers, so it is impossible to make use of the data in their system. However, their General Transit Feed Specification provides a good starting point as to what data is needed in order to make a good public transport system[10]. What differs Google Transit from the other systems looked at in this survey is that Google Transit is a purely server based system. All business logic resides on the server. The client, which is a web browser, only handles the display of information. Having the business logic on a server provides a lot of benefits for the solution to be created as well. One of the benefits is that it saves the client for a lot of work. This is desirable because it saves CPU cycles, which consequently saves battery. This approach finds support in other works. For instance in an article by Forman and Zahorjan (1994) where they draw the very obvious connection between CPU cycles and battery power consumption[6]. Wireless data transfer should be considered as well. If all business logic was to be handled on the client side, due to extraneous factors, more data has to be transferred to the client. This

[1] The 'U' in FUIROUS

is because our the business logic requires several request to a SOAP[42] service for real-time data[43], and one request towards BusTUC. By handling the business logic on the server, the server can make these request to the various services. Since some of the requested resources can be shared by the clients using the server, the server can also relieve the external services for heavy request load. Regarding shared resources, there is another aspect to consider. Since there is a single point where the logic is handled, the possibilities for optimisation trough resource sharing and information caching is easy to implement. A single point where the logic is handled can also be exploited in other ways. For instance, if one of the external service providers decides to alter how their service is accessed, only one update is needed. With business logic implemented on a server, there is only one point where the code has to be changed. If the business logic existed on the client, all applications had to have their code updated to handle the service change.

All this research points in the direction of the hybrid strategy, using HTML5, javascript, CSS and a deployment technology to easily create a client prototype that works on several platforms. The client prototype should consist of all the functionality in Magnus Rauum's application. This includes a search function and a map view. Also, the ability to use bookmarks or favourites should be implemented. As described a server also offers many benefits. Therefore, our solution consists of two parts, the server and the client. Together the server and client implements functionality needed to take the next step into becoming the ultimate bus route system.

# 4 Prototype Client

This chapter gives an overview of promising frameworks and what they can contribute in the context of mobile application development. First there is be a brief in introduction to the basic technologies that most of the available frameworks use.

## 4.1 Development Technologies

This subsection describes the languages of web application development used in this project.

[42] `http://en.wikipedia.org/wiki/SOAP`

[43] `https://www.atb.no/sanntid/category210.html`

### 4.1.1 HTML 5

HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG). HTML5 will be the new standard for HTML, XHTML, and the HTML DOM. The previous version of HTML came in 1999. The web has changed a lot since then and new functionality is needed to give more HTML native support for new functionality. HTML5 is still work in progress. However, some rules for the final HTML5 standards is established:

1. The new features should be based on HTML, CSS, DOM, and JavaScript
2. Reduce the need for external plugins (like Flash)
3. Provide better error handling
4. Contain more markup, to replace scripting
5. HTML5 should be device independent
6. The development process should be visible to the public

The new main features in HTML5 are:

1. Canvas element for drawing
2. Video and audio elements for media playback
3. Better support for local offline storage
4. New content specific elements, like article, footer, header, nav, section
5. New form controls, like calendar, date, time, email, url, search

The most interesting new feature in the context of this project, is better support for local and offline storage. With the old HTML standard the only way to save information directly on the client is *cookies*. Storing information in cookies is very inefficient, because cookies are loaded on every server request. Since this project evolves around mobile devices, it is imperial that data transfer is kept to a minimum. A natural way to exploit the new functionality in this project would be to download as much as possible of the needed data, first time the application is used on a mobile device. Typically this data could be bus-route-cross-reference tables.

### 4.1.2 CSS 3

Cascading Style Sheets is a style sheet language used to describe the presentation semantics of a document written in a markup language, like XML. CSS is primary designed to make it easy to style fonts, color and layout for different parts of an webpage.

The new CSS3 standard differs from the old ones in that it uses modules that handle different types of styling. These modules are manifested text documents, and each module adds new capability or extends features defined in CSS 2 standard. The first CSS 3 draft came already in June 1999, but the the first W3C recommendation for a CSS 3 module was made in in June 2011.

Later in this report one can see that some of the most promising JavaScript frameworks are using CSS 3 for styling purposes.

### 4.1.3 JavaScript

JavaScript is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions, as explained below. It is a multi-paradigm language, supporting both object-oriented, imperative and functional programming styles. Prototype-based simply means that one does not use classes. Behaviour reuse is accomplished through cloning of existing objects which then serves as prototypes. "Weakly typed" means that JavaScript is not strict on how different data types are mixed. JavaScript has first-class functions, which means that it treats functions as first-class objects, and can therefore pass function as arguments to other functions.

JavaScript was a for long time seen as the black sheep of the web, but since the AJAX web development method became popular JavaScript has redeemed itself. Traditionally JavaScript has only been used on the client sides, but lately better virtual machines has been developed for it to run on. Therefore, JavaScript is now also on the server side.

As one can see from the HTML 5 section above, JavaScript plays an important role when working with the new HTML standards. As JavaScript is a scripting language, it makes it possible to move most of the business logic to the client side of the web-application, if desired.

Several JavaScript frameworks out there helps to ease the pain of making a graphical user interface. Two of these, "Sencha Touch" and "JQuery Mobile", are popular alternatives and provides everything we need for our application:

**Sencha Touch** Sencha Touch[44] is a cross-platform framework aimed at the next generation, "touch enabled", devices. It is currently compatible

[44] `http://www.sencha.com/products/touch`

with Apple iOS 3+, Android 2.1+, and BlackBerry 6+ devices. Sencha Touch is the world's first app framework built specifically to leverage HTML5, CSS3, and Javascript.

**JQuery Mobile** JQuery Mobile[45] is a unified, HTML5-based user interface system for all popular mobile device platforms, built on the rock-solid jQuery and jQuery UI foundation. Its lightweight code is built with progressive enhancement, and has a flexible, easily "themeable", design.

## 4.2 Deployment Technologies

There are several available mobile development frameworks that are capable of hybrid cross-platform application deployment. These frameworks can free developers from having to write any code in the target devices' native languages. Developers can stick to one codebase, but still get access to many of the device's native features, such as the compass, the camera, the contact list, and so on. Three popular frameworks, with different strengths and weaknesses, have been reviewed in order to choose the right one to use.

### 4.2.1 PhoneGap

PhoneGap[46] is an open-source mobile development framework that enables software programmers to build applications using the "standard web stack" (JavaScript, HTML5 and CSS3). The resulting applications are hybrid, neither fully native nor purely web based. This provides the ability to make an application as complex as needed, but still have the opportunity to let it degrade in order to work on devices with less support for new content. This degradation can be controlled via CSS, or even dynamically with JavaScript by checking the type and version of browser the device has installed and what content this browser supports. PhoneGap does not limit development to any specific integrated development environment (IDE[47]) or framework. Developers may choose whatever tools they like. This makes PhoneGap ideal for cross-platform development. It does have its backsides, though. The applications created will have its layout rendered in a webview, which is slow in comparison to a native GUI on the device. The performance will be better in the near future, when support for graphics processing unit (GPU) acceleration is implemented on all mobile devices. Even though hybrid applications have access to the smartphone's utilites such as sensors, camera and contacts, it does not have full access to the device application programming interface (API). PhoneGap provides a

[45] `http://jquerymobile.com/`

[46] `http://www.phonegap.com/`

[47] `http://en.wikipedia.org/wiki/Integrated_development_environment`

web service called "PhoneGap: Build"[48] that translates the HTML5-code to native code for all the platforms it supports: iOS, Android, webOS, Symbian and BlackBerry. Support for Windows Mobile is still in development.

### 4.2.2 Appcelerator Titanium Mobile

Appcelerator Titanium[49] is another mobile web development application framework. Appcelerator is similar to Phonegap in that it tries to pursue the notion of cross-platform development. It differs from PhoneGap in that deployment is limited only Android and iOS platforms. While writing in familiar JavaScript syntax, developers will also have to learn the extensive Titanium API, and to use designated tools provided by Appcelerator. These tools are free. Using the Titanium API is both a good and a bad thing. Learning the API requires a steeper learning curve than more familiar web frameworks, and because it is purely javascript and has no ties to the DOM, it cannot make use of opular javascript libraries. Most of them, like JQuery, require the presence of a DOM window and document. The Titanium API does give us the ability to implement native UI components on the iOS and Android platforms. This is where the Appcelerator shines. Where the PhoneGap apps, using webview, will seem a bit sluggish in animations and GUI interaction, the Appcelerator applications look, feel and perform like native applications. Note that this is after the application is loaded, since the loading times could actually be slightly slower. This is because the Appcelerator is not truly a cross-platform compiler. What really happens, after the application source code is deployed to the mobile devices, is that it is interpreted every time the application runs. In addition, errors in the source code do not reveal themselves until run-time. This could be time consuming to debug during development.

### 4.2.3 Rhodes

Rhodes[50] is a mobile web development framework that uses a Model View Controller (MVC)[51] pattern. The views are written in HTML and the controllers in Ruby. Rhodes supports the iOS, Android, Windows Mobile, Blackberry and Symbian platforms. Rhodes compiles to true native applications, which gives the same strengths as for the Appcelerator Titanium. Rhodes is powerful and feature-rich, but does not have the simplicity and ease of use that Titanium offers. Significant specialist knowledge is required for good results. Rhodes tries to manage the entire application development from start to end. This may put obstacles in the actual process of

[48] `https://build.phonegap.com/`
[49] `http://www.appcelerator.com`
[50] `http://rhomobile.com/`
[51] `http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller`

making an application for some developers. If something goes wrong anywhere in the development process, e.g. compilation errors, or one want to do something in a different way than the Rhodes common practice it may prove difficult to do. Rhodes, just like PhoneGap, also supports automatic codebase deployment to different platforms, through RhoHub[52].

### 4.3 Conclusion: Deciding which Technologies to Make Use of

Since one of the prerequisites in this project was to reach out to as many mobile device platforms as possible, it is the key point when choosing a deployment technology. Appcelerator Titanium's lack of support for any platforms except iOS and Android makes it a poor alternative. As for Rhodes, it does have full platform support, but the requirement for knowledge of Ruby and the extensive API will perhaps make it a job too big to handle for this size of project. Another prerequisite is that the framework should be easy to use and be able to collaborate with other frameworks that can make it faster and easier to create well working GUI-components. This is easier in PhoneGap than in the other deployment technologies. The Multi-BRIS application in this project does not require very heavy computing or graphics, so the performance weakness of PhoneGap is not a big problem. PhoneGap's simplicity and platform support therefore makes it a clear winner for our project.

Regarding which JavaScript framework to use, the updated (February 17, 2011) article "JQuery Mobile vs Sencha Touch"[53] by Tyson Lloyd Cadenhead gives a concise overview over the two frameworks and for what sort of developers and projects they are prefered. The main difference between the Sencha Touch library and JQuery is that Sencha Touch rarely use any pre-rendered HTML and the developer typically work with components that consists of several DOM elements bound as one object instead of working directly with the DOM, one element at a time. The authors of this report are primarily Java developers and do not have extensive knowledge of coding webpages in HTML, so Sencha Touch's near pure JavaScript approach and sense of control makes this our best choice.

## 5 Prototype Server

This chapter describes the prototype server developed during this project. First, a system overview and a description of the technologies is given. Then each service is described in more detail.

[52] `https://app.rhohub.com/`

[53] `http://tysonlloydcadenhead.com/blog/jquery-mobile-vs-sencha-touch/`

## 5.1 System Overview

The server offers three distinct services. Figure 11 shows a block diagram of the system.

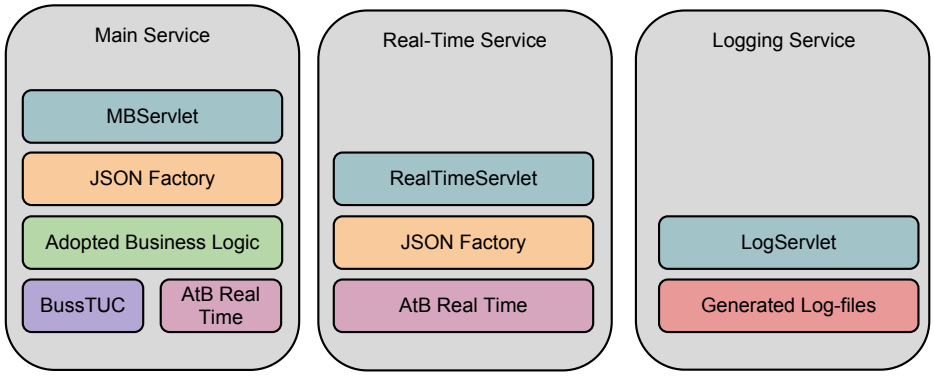


Figure 11: Block diagram of the server system, MBServlet: MultiBRIS servlet, JSON Factory: JavaScript Object Notation Factory.

The main service effectively replaces all the business logic implemented by Magnus Raaums[13] Android application. The client sends its desired destination and the current location to the MultiBRIS server. The server looks up the desired number of bus-stops near the given location. When the bus-stops are found, a query is sent to the new BusTUC web-interface (the new BusTUC web-interface is described in section 6). The new BusTUC web-interface responds with possible bus routes. The MultiBRIS server then updates these routes with real-time departure times for the buses proposed by BusTUC. This is done by contacting a real-time system provided by AtB[54]. The MultiBRIS server then sends the updated route alternatives back to the client.

The system also offers a pure real-time service where the client can send a bus-stop ID to the MultiBRIS server and get a list of the next five buses arriving at that stop. AtB offers a similar service directly, but with some significant drawbacks. These drawbacks are elaborated in the discussion section (Section 8).

When the first version of the server was up and running, it was quickly established that a easily accessible logging system was needed for debugging reason. This was accomplished through making a logging service, which is accessible trough a web browser. The service was made so that it is easy to change the amount of data logged.

[54] `https://www.atb.no`

## 5.2 Technologies

The server technology used is Java Servlet[55], developed by Sun Microsystems in 1997. The Java servlet technology is now at version 3.0 and has by now been around for over a decade. It is a well tested and documented technology. With Java Servlets, business logic can easily be written in Java[1] and then be made available to consumers through servlets[12]. As the starting point for the project was to make a multi-platform client, it was natural that the server was created with the same though in mind. A Java servlet can be published through any available servlet container, which makes very portable. A servlet container, also known as a web container, is a component of a web server that interacts with a servlet and makes the communication between a servlet and a web-client possible. There are many servlet containers available. Some are commercial and some are not. The best known commercial containers are IBM's WebSphere[56] and SAP's NetWeaver[57]. Among the non-commercial servlet containers are Apache's Tomcat[58] and Glassfish from Sun Microsystems[59]. As this was an academical project a non-commercial server would be needed. The one chosen is called Jetty, from Eclipse Foundation[60], it is seen at a rising star in the web community, much because Google chose to use Jetty technology for there cloud service, Google App Engine[61]. What is special about Jetty is that it is made up of pure Java code. As known Java is very portable[1]. This all facilitates a very portable server solution.

## 5.3 The Main Service

This is the service that implements all of the business logic needed to replicate Magnus Raaum's application[13] as a multi-platform system.

### 5.3.1 Migrating the Application Logic to the Server

When migrating the business logic to the server, major changes was done. The previously logic was used on an Android device, utilising Android- and Google libraries. The new business logic needed to be made independent from these. Another thing that had to be sorted out was to organise the code, as the previous structure had too much functionality nested together. For instance the functionality for handling request to an external

[55] `http://www.oracle.com/technetwork/java/javaee/servlet/index.html`
[56] `www.ibm.com/software/websphere/`
[57] `http://www.sap.com/platform/netweaver/index.epx`
[58] `http://tomcat.apache.org/`
[59] `http://glassfish.java.net/`
[60] `http://www.eclipse.org/jetty/`
[61] `http://code.google.com/intl/no/appengine/`

service and sorting routes where in the same Java method. It was also vital to do proper exception handling and logging. The way the system was implemented meant that if some part of the code would lead to an exception when running, the code would halt. This could typically be if BusTUC returns an erroneous answer. Then the system would not proceed to the ranking involving real-time data. This, in turn, would lead to no proper error message being sent to the client side.

One of the main issues when breaking the connection to the Google API was that the system used Google functionality to calculate distance between GPS coordinates. This functionality had to be completely reimplemented when moving the logic to the server. Calculating the distance between two GPS locations is not as easy as just to use euclidean distance, because of the approximately oblate spheroid[62] shape of the earth. Therefore, the reimplemented logic used to calculated the distance between two GPS locations is now based on the "Inverse formula" from a paper by T. Vincentry (1975), using the WGS 84 standard[63][16].

### 5.3.2 Interacting With BusTUC

The BusTUC natural language system accepts two forms of queries. One is natural language through sentences, and the other is a nested query. Both are shown below.

- When does the bus depart from Ilsvika to Tiller?
- (Ilsvika +$n$, Ila +$n$) til Tiller.

MultiBRIS uses only on the second type of query. Its syntax allows nested queries with multiple departure stops. The answer from BusTUC is also different when using the second query type. In addition to a textual answer in the HTML returned, an JSON object can be filtered out. The textual answer is actually never used, as the JSON object is what is parsed for information. The $n$ represents "walking distance to", in this case Ila and Ilsvika. The $n$ parameter was discovered to not work at all during testing, as different values did not affect query results.

### 5.3.3 Main Service Interface

A query is posed to the server as a HTTP GET, and parameter data is sent trough the URL. A typical query to the server looks like this: `http://bustjener.idi.ntnu.no/MultiBRISserver/MBServlet?dest=Ilsvika&type=json&lat=63.4169548&long=10.40284478&nStops=5&maxWalkDist=300&key=TheKey`

[62] `http://en.wikipedia.org/wiki/Oblate_spheroid`
[63] `http://en.wikipedia.org/wiki/WorldGeodeticSystem`

In table 3 one can see a complete list of possible attributes and what their function is. How many concurrent request the server is allowed to handle is regulated by the servlet container, in this case Jetty. Jetty has an configuration file with a property called thread pool, where the number of allowed concurrent requests can be regulated.

| **Attribute** | **Function** |
|---|---|
| dest (required) | The desired destination. |
| type (required) | Can be aether *json* or *text* depending on desired return type. |
| lat (required) | Latitude value for current location. |
| long (required) | Longitude value for current location. |
| nStops | Number of bus-stop to check for possible routes (Default is 5). |
| maxWalkDist | Max distance in meters to checked bus-stops (Default is 500). |
| key | Key is used to identify users of the system (Default is null). |
| callback | Adds callback part to the return with the given callback value. Used for HTML injections to bypass same origin policy restrictions for AJAX. |
| getSpecialTransfer | Changes the return so that the two or more routes involved in the transfer appear as one alternative. |

Table 3: List of attributes and their functions for the main service.

A typical return from the server would look like the one in listing 1:

```
{ "alts" : [ { "arrivalTime" : "1317",
        "busNumber" : 8,
        "bustopName" : "Prof. Brochs gate",
        "bustopNumber" : 16011376,
        "depLatitude" : 63.415534999999998,
        "depLongitude" : 10.398126,
        "destination" : "Torget",
        "totalTime" : 1322,
        "transfer" : true,
        "travelTime" : "05",
        "walkingDistance" : 283
      },
      { "arrivalTime" : "1334",
        "busNumber" : 63,
        "bustopName" : "Dronningens gate - D1",
        "bustopNumber" : 16010006,
        "depLatitude" : 63.432020000000001,
        "depLongitude" : 10.392315,
        "destination" : "Ilsvika",
        "totalTime" : 1341,
        "transfer" : true,
```

```
        "travelTime" : "7",
        "walkingDistance" : 1759
      }
    ],
  "error" : false,
  "formatedTexts" : "1: Ta bus 8 fra Prof. Brochs gate (283
     meter) klokken 1317. Du vil nÃ¥ Torget ca 5 minutter
     senere.\n2: Ta bus 63 fra Dronningens gate - D1 klokken
     1334. Du vil nÃ¥ Ilsvika ca 7 minutter senere.\n"
}
```

Listing 1: JSON return from server.

Figure 12 shows a complete interaction diagram for the main service.

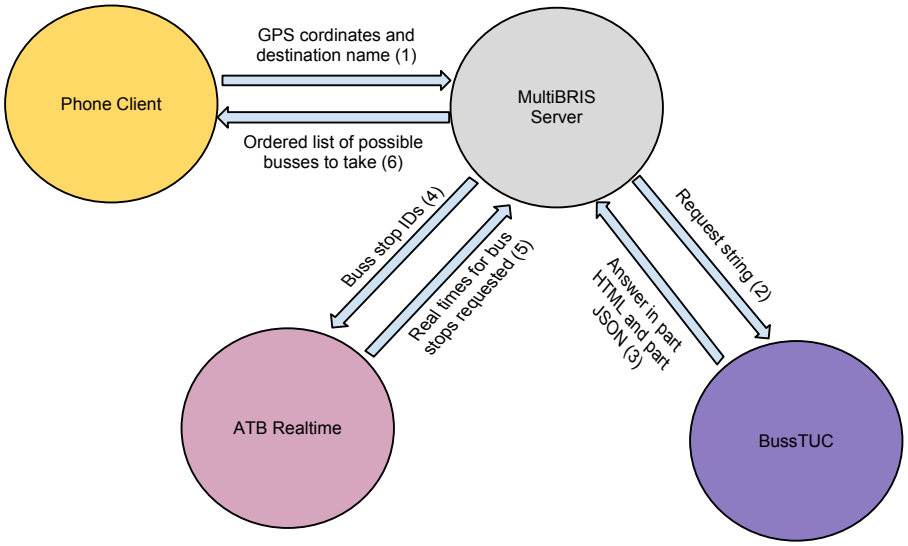


Figure 12: Main service overview.

## 5.4 The Real Time Service

A client can access the real-time service by using a HTTP GET against the MultiBRIS server. The request URL looks like this: `http://bustjener.idi.ntnu.no/MultiBRISserver/RealTime?bID=16010495`
The table 4 shows the possible attributes for this service. A typical return from the real time service would look like the on in listing 2.

```
{ "busStops" : [ { "arrivalTime" : "27/11/2011 15:25:00",
        "dest" : "Klabu sentrum",
        "line" : 47,
        "realTime" : false
      },
      { "arrivalTime" : "27/11/2011 16:25:00",
```

| Attribute | Function |
|---|---|
| bID (required) | The Bus-stop ID for which to return bus arrivals for. |
| callback | Adds callback part to the return with the given callback value. Used for HTML injections to bypass same origin policy restrictions for AJAX (For details see section 8.1.3). |

Table 4: List of attributes and their functions for the real time service.

```
      "dest" : "Klabu sentrum",
      "line" : 47,
      "realTime" : false
    },
    { "arrivalTime" : "27/11/2011 19:25:00",
      "dest" : "Klabu sentrum",
      "line" : 47,
      "realTime" : false
    },
    { "arrivalTime" : "27/11/2011 20:25:00",
      "dest" : "Klabu sentrum",
      "line" : 47,
      "realTime" : true
    },
    { "arrivalTime" : "27/11/2011 21:25:00",
      "dest" : "Klabu sentrum",
      "line" : 47,
      "realTime" : false
    }
  ] }
```

Listing 2: Real time JSON return from server.

Figure 13 shows a interaction diagram for the the real time service.

## 5.5 The Logging Service

To understand how logging works in MultiBRIS, a little info about how Java Servlets works in relation to threads is needed. If nothing else is specified, a Java Servlet creates a new thread for each new client session. In MulitBRIS, this thread is given a unique recognisable id. The id is created by combining the value of the key attribute from table 3 and the value from the System.nanoTime()[64] method in Java. This gives the threads a unique id and at the same time it is easy to track the threads in a multi thread system by the key part of the thread id. The reason for this thread tracking in relation to logging has to do with the design of the logger. The logger was designed so that it takes a minimum amount of work to use it anywhere in the code. At the same time it makes use of only a single log object for each

[64] http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html

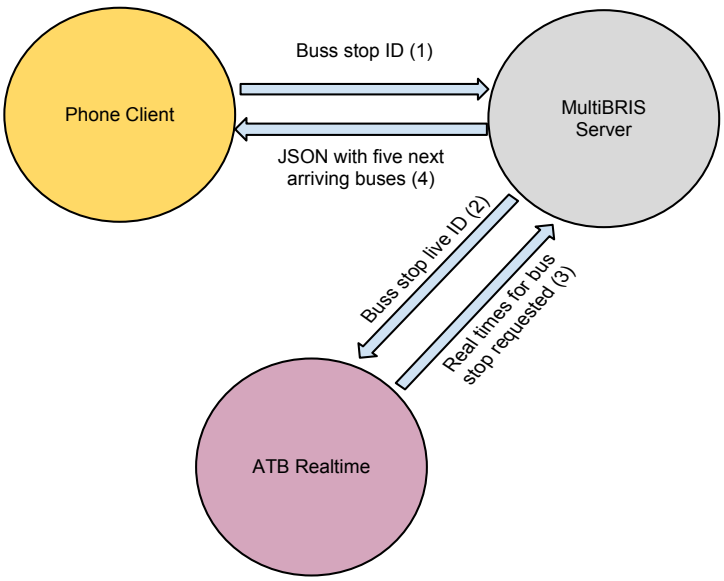


Figure 13: Real time service overview.

service. The normal way of doing this in Java would be to send the logger object as an parameter to each method. That however, is not a very elegant method of implementing it. Instead the logger is made as an Java singleton[65] and fetched when ever needed anywhere in the code. The logger then uses the thread name of the caller to identify the user session. That way, it is possible to have a single logging object that can be used by all parts of the system, and still let the logger keep track of who the caller is.

The file writing is handled by Java's built in functionality, java.util.logging[66], but a custom file handler was made to make log entries compatible with HTML representation. The log is accessible trough a web service, a client can access the logs by doing ha HTTP GET request like this: `http://bustjener.idi.ntnu.no/MultiBRISserver/MBServlet?getLog=true#bottom`

Figure 14 shows some example log entries returned trough the web service. One can see from the $qID : TheKey23411709$ that both these log entries are from the same user session. The log level is in this example set to fine, meaning it is very verbose, one can see that both entries with log level "INFO" and "SEVERE" are present. In a production scenario the log level would be changed to "SEVERE", so that only the system errors would be logged. As there is only a single instance of the logger, the change would have to be done only one place in the code.

[65] `http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-designpatterns.html`

[66] `http://docs.oracle.com/javase/1.4.2/docs/api/java/util/logging/package-summary.html`

| | |
|---|---|
| INFO | Dec 04,2011 12:46 qID: TheKey23394413 BUSTUCSTR wanted_string:(Gløshaugen Nord+3,Gløshaugen |
| SEVERE | Dec 04,2011 12:46 qID: TheKey23394413 IO EX e:org.apache.http.conn.HttpHostConnectException: |
| INFO | Dec 04,2011 12:46 qID: TheKey23394413<br>Key: TheKey<br>queryID: TheKey23394413<br>Remote host: 129.241.107.25<br>Remote address: 129.241.107.25<br>User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:8.0.1) Gecko/20100101 Firefox/8.0.1<br>Lat: 63.4169548<br>Long: 10.40284478<br>dest: Ilsvika<br>nStops: 5<br>maxWalkDist: 300 |

Figure 14: Example log return.

### 5.6 Server Optimisation

For optimisation of the MultiBRIS server two things were done. The "bus ID to bus-stop live ID" lookup list are shared between all the clients using the server. Threading was introduced in the retrieval of real-time data for multiple bus-stops. This resulted in a computation time half of the original, as multiple threads now send queries in parallel, instead of sequential. However, the time between the query was posed to the server and the server responded was still too long. The query time was still sometimes up to 40 seconds. The reason for this was that the MultiBRIS server had to wait for answers from the BusTUC server. As a response to this it was decided that speeding up the BusTUC server was imperative for the practical usability of the entire system. Therefore a new web-interface for BusTUC was made, as described in section 6.

## 6 New Web-Interface for BusTUC

This section describes the new BusTUC web-interface developed during this project. First, challenges related to the previous web-interface is explained. Then the new solution is presented.

### 6.1 Challenges with the Previous BusTUC Web-Interface

The main challenge with the previous solution was the lack of similar request handling. If the server was already handling a request the next request would just have to wait for the current running request to finish. If the first request takes more than 30 seconds the next request is dropped. This behaviour is produced by the way the web-interface is implemented. It uses a PHP-script that writes to a predefined file when a request is posed to the script. The BusTUC back end system then runs a service that checks this file once every second. If there is new content in the file, it reads the content from the file and processes the request. BusTUC then writes to another predefined file which the PHP-script, in turn, checks for new content

once every second. This method of exchanging information, even if not taking into account the extra time used to write and read from files, has a two second potential delay, because of the checking intervals.

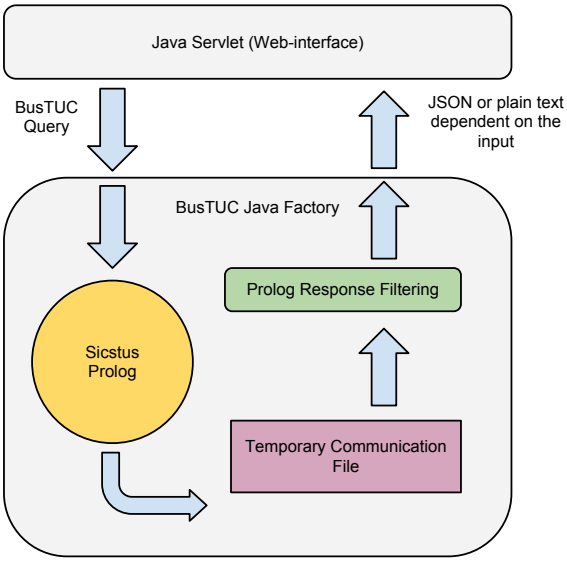


Figure 15: New Web-Interface Overview.

## 6.2 The New Web-Interface

The new BusTUC web-interface uses Java Servlet technology with the same Jetty container as the MultiBRIS server. This is because it is well documented and portable. The BusTUC core system is made of Prolog code, which is a general purpose logic programming language [67]. The specific Prolog framework used is called Sicstus and is maintained and owned by the Swedish Institute of Computer Science[68]. Sicstus has made a java library called Jasper[69] which makes it possible for Java code to communicate with Prolog code. Figure 15 shows an overview of the new web-interface.

When a query arrives at the Java Servlet the query is sent to what is called the BusTUC Java Factory. The BusTUC factory poses the query to the Prolog code, which, in turn, puts the answer into a temporary file. What happens in the Prolog part of this factory is considered out of scope for this project. Thus it is treated as a black-box[70]. The reason why the exter-

[67] http://en.wikipedia.org/wiki/Prolog
[68] http://www.sics.se/isl/sicstuswww/site/index.html
[69] http://www.sics.se/sicstus/docs/3.7.1/html/sicstus_12.html
[70] http://en.wikipedia.org/wiki/Black_box

nal communication file is needed is because the BusTUC system was not made to be interfaced the way it is in this project. To be able to exclude the usage of this file, approximately 1000 lines of Prolog code would have to be changed according to an external resource. Therefore, the usage of a file as intermediate layer was accepted as a necessary evil. The filename is made unique by using the thread name and the current system nanotime. The file is then read and filtered by another part of the the BusTUC factory before the result is returned to the Java Servlet. A typical request to the oracle where the goal is to get a JSON object back looks like this: `http://furu.idi.ntnu.no:1337/bussstuc/oracle?jq=%28H%F8gskoleringen%2B1,Gl%F8shaugen%20nord%2B1,Gl%F8shaugen%20syd%2B1,samfundet%2B1%29%20til%20dragvoll`
The return looks like this:

```
 "departures" : [ { "busnumber" : 5,
       "busstopname" : "GlÃ¸shaugen Nord",
       "busstopnumber" : 16010333,
       "destination" : "Dragvoll",
       "duration" : 9,
       "time" : 2348
     },
     { "busnumber" : 5,
       "busstopname" : "Studentersamfundet",
       "busstopnumber" : 16010477,
       "destination" : "Dragvoll",
       "duration" : 13,
       "time" : 2404
     },
     { "busnumber" : 5,
       "busstopname" : "HÃ¸gskoleringen",
       "busstopnumber" : 16010197,
       "destination" : "Dragvoll",
       "duration" : 11,
       "time" : 2406
     },
     { "busnumber" : 5,
       "busstopname" : "GlÃ¸shaugen Syd",
       "busstopnumber" : 16010265,
       "destination" : "Dragvoll",
       "duration" : 9,
       "time" : 2408
     }
   ],
  "timeset" : "false",
  "transfer" : "false"
}
```

Listing 3: JSON return from new BusTUC Web-interface.

If the normal text answer is desired, a request would look like this : `http://furu.idi.ntnu.no:1337/bussstuc/oracle?q=ilvika%20til%20gl%F8shaugen?`
The return would look something like this: *Holdeplassen nærmest Gløshau-*

*gen er Gløhaugen Syd .Buss 63 går fra Ilsvika kl. 2325 til Munkegata M3 kl. 2331 og buss 52 går fra Munkegata M3 kl. 2350 til Gløshaugen Syd kl. 2357 .Tidene angir tidligste passeringer av holdeplassene.*

# 7 Results

The results section presents the quantifiable results produced by this project.

## 7.1 Physical Servers

There are two physical servers used in this project, they are called busstjener.idi.ntnu.no and furu.idi.ntnu.no. The server busstjener.idi.ntnu.no provides the MultiBRIS service and furu.idi.ntnu.no provides both the new and the old BusTUC web-interface. All results involving a "back-end" service, except AtB's real-time service, are produced using these servers. In table 5 and table 6 the specification of the servers are given.

| **Attribute** | **Value** |
|---|---|
| CPU | 2x 5.2 GHz, VMware shared pool [71] |
| Memory | 4 GB dedicated |
| OS | Ubuntu 11.04 (GNU/Linux 2.6.38-8-server x86_64) |

Table 5: Server information for busstjener.idi.ntnu.no.

| **Attribute** | **Value** |
|---|---|
| CPU | 4x UltraSPARC IIIi 1.062, 1.28, 1.593 GHz |
| Memory | 16 GB dedicated |
| OS | Sun Microsystems Inc. SunOS 5.10, Generic January 2005 |

Table 6: Server information for furu.idi.ntnu.no.

## 7.2 The Client Application

This section shows a number of figures depicting the functionality of the MultiBRIS client application. Note that when the application starts it tries to retrieve the current location. If it fails, the application is basically useless because all functionality is based on this information.

**Figure 16:a** This is the Favourite tab. Here the user can add favourites by tapping the "+" button (See figure 16:c) or remove them either by swiping the favourite and tap the "X" button or mark the favourite and tap the button that looks like a trash can. Tapping any of the favourites that is added

Table 7: Translation of menu elements

| Norwegian | English |
|---|---|
| Søk | Search |
| Favoritt | Favourite |
| Destinasjon | Destination |
| Lagre | Save |
| Avbryt | Cancel |
| Kart | Map |
| Meg | Me |
| Nær meg | Near me |
| Fjern | Clear |
| Resultater | Results |
| Oppdater | Update |

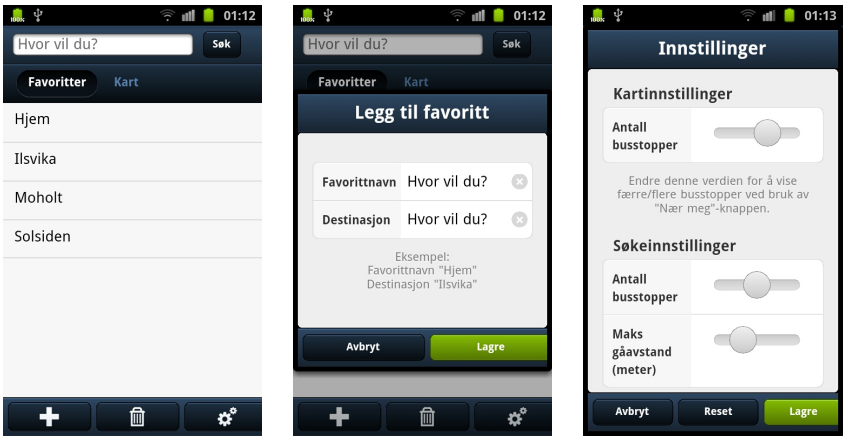


Figure 16: (a)Favourite tab, (b)Add favourite and (c)Settings.

immediately starts a bus route search for that favourite's destination and the user is presented with the Results tab (See figure 17:b).

**Figure 16:b** This figure represents the add favourite feature where the user types in the name and destination of favourite.

**Figure 16:c** This is the settings view where the user can set properties for how many bus-stops should be viewed in map mode or how many bus-stops should be assessed in the bus route queries.

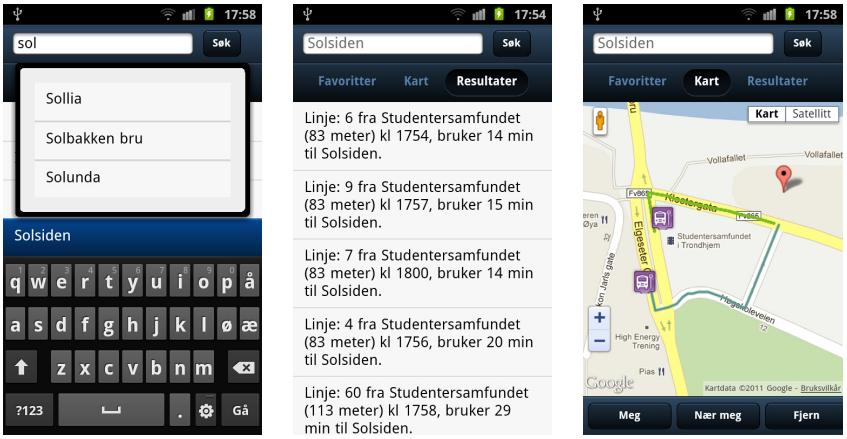


Figure 17: (a)Searchbar, (b)Results tab, (c)Results shown on the map.

**Figure 17:a** A search bar always resides on top of the application. This gives the user quick access to bus route search functionality from wherever the user has navigated in the application. The search bar has an autocomplete function that suggests bus-stops that exist in Trondheim. The rest of the figures represent the views that are available through the tab mechanism.

**Figure 17:b** This is the result tab. Here the user is presented with the (up to) 5 most optimal bus routes for the search query. They are presented in a list, sorted by total travel time. The user can now either tap on the result that suits the most to switch to the map and view the target bus-stop and travel route to that bus-stop, or the user can click on the map tab and be presented with all the bus-stops in his results along with the travel routes to them from his location.

**Figure 17:c** After doing a search, the user may click on the map tab in order to get a view of the bus-stops that are among the results. Also, colored lines shows the user how to get to them from the user's current location.

**Figure 18:a** The figure depicts the real-time information view. It shows information of the next 5 buses for specific bus-stop. Buses marked in red, have actual real-time values, while the black are regular times. If any of the buses are among the users search results, they are also marked with a "*".

**Figure 18:b** This figure describes the map tab of the application, consisting of a Google Map with added functionality. Here the user is presented with the current location centered on the map. The user can at any time update or pan back to a location by tapping the "Meg"-button.

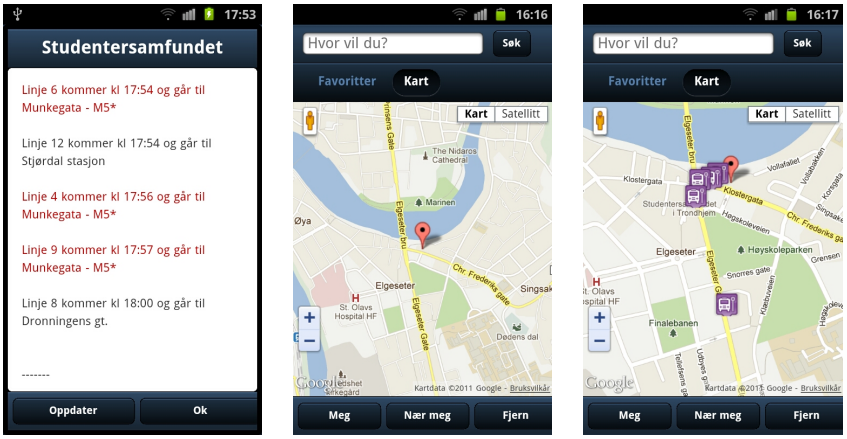


Figure 18: (a)Real-time information, (b)The Google Maps, (c)Close-by bus-stops functionality.

**Figure 18:c** In the map tab the user can also click on the "Nær meg" to add closeby bus-stops to the map. The user may then click on any of these to get real-time information of the 5 next buses passing through that bus-stop (See figure 18:a).

## 7.3 The Server

The easiest quantifiable result of moving the business logic to a server is the decrease in data transfer. Table 8 show a comparison between the data transfer for the client, before and after the business logic was moved to the server. The measurement scenario consists of starting the client and making a bus route alternatives query i.e. using "the main service" on the MultiBRIS server, as described in section 5.3. This scenario was chosen as it would represent the most natural usage pattern. As one can see, moving the business logic to the server resulted in that only a 100 fraction of the original data transfer amount is needed for a normal user scenario. About 400 KB of the transferred data comes from downloading the live ID to bus-stop ID list from AtB.

| **Business logic on client** | **Bussines logic on server** |
|---|---|
| 570 KB | 5 KB |

Table 8: Average data transfer comparison for bus route query (Measured with WireShark 1.6.1. The Client used was a AppleWebKit 535.2 Based Browser.)

When it comes to saved CPU cycles it is not easy to measure the result of

moving the business logic. No known mobile devices have functionality for this. However if one instead looks at the main incentive for saving CPU cycles, namely power usage, there are some tools available. In figure 19 there is a comparison between power usage for having business logic on the client and on the server side. The topmost graph in figure 19 shows the total amount of milliwatts (mW) usage for the application. As one can see the one where the business logic is on the client use more power.

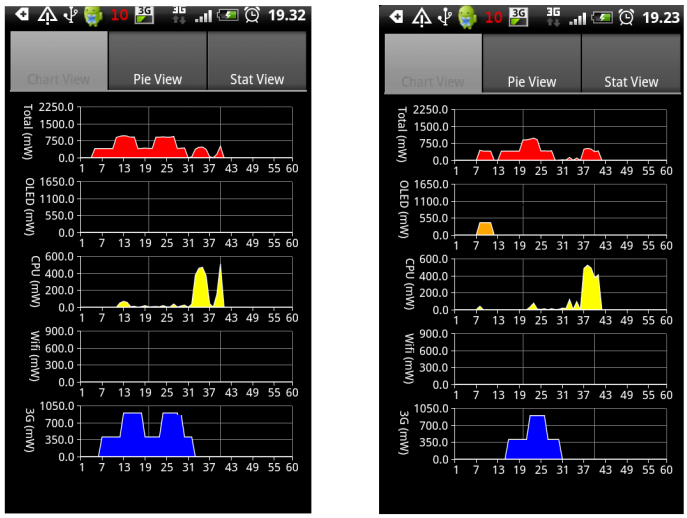


Figure 19: Power usage with business logic on client (left) and business logic on sever (right). Test Client Provided by the TABuss project.

## 7.4 The New BusTUC Web-Interface

As mentioned it was imperative that the BusTUC query time was reduced and as we can see in table 9 a substantial reduction in query time was achieved. The most important improvement is the max query time, which was lowered to one third of the old web-interface.

| | **Old Web-Interface Query Time** | **New Web-Interface Query Time** |
|---|---|---|
| Max | 30 sec | 10 sec |
| Avg | 15 sec | 6 sec |

Table 9: BusTUC Query Times.

# 8 Discussion

This section discusses the results from the Result section and the challenges that was encountered during development.

## 8.1 Challenges During Development

This subsection describes a variety of challenges that appeared during development of the prototypes, both on server and client.

### 8.1.1 Bugs in the Previous Business Logic

During development, some bugs were found in the existing business logic. The following list contains the critical errors that was found and corrected.

1. When the total travel time was equal for more than one route, due to how Java HashMaps works, routes was replaced during sorting and an error occured in later computations because of missing items.

   This bug was fixed by not using the total time from the HashMaps as sort parameters.

2. When the distance to two bus-stops was equal, the latter bus-stop added to the system overwrote the first because the distance was used as key parameter in the HashMap that holds the bus-stops. This bug did not give an exception under system execution, but it removed a bus-stop that might have been part of the ultimate bus route alternative.

   This bug was fixed by making sure distances never were the same. This is easy to do when dealing with HashMaps as no actual search is needed to check if the key is unique. One can simply pull out an object from the HashMap using the key in question. If the returned object is null, then all is good. If not, increment the key and try again. Of course this could make one of the distances move off by a metre or two, but this does not really matter because the distances are straight lines that do not take terrain into account, making them inaccurate anyhow.

### 8.1.2 AtB's Real-Time Service

To understand the challenge with AtB's real time SOAP[72] service one need to have an overview of the information entities involved. A bus-stop essential has four info entities in this system; name, ID, GPS location and live

[72] `http://www.w3.org/TR/soap12-part1/#intro`

ID. The live ID may change whenever AtB's real-time system is updated or is restarted. The live ID is not updated often by AtB's system, but it has to be update relatively often by the consumer of the service to make sure bus-stop data is correct at all times. Because the consumer of the service never knows when the live IDs are changed. The bus-stop's live ID is needed to give the five next arriving buses to the queried bus-stop. The problem then, is that it is not possible to query AtB's real-time SOAP system for one live ID, given a bus-stop ID. It is however possible to query the service for a complete list of bus-stop IDs and live IDs, which is a big chunk of data. The SOAP service also has some bugs; it changes the returned SOAP message sporadically, so that it does not match the expected return given in the WSDL [73]. In addition, the SOAP service's parameters are all in Italian, which makes it troublesome to use without knowledge of Italian. Therefore we made a mediation layer that simplifies the query for a client by removing the need for handling bus-stop ID to live ID translation, strange SOAP returns and Italian parameters. This service is described in section 5.4

### 8.1.3 Same Origin Policy

The same origin policy[74] is an important security concept for a number of browser-side programming languages, such as JavaScript. The "origin" term describes resources having the same application layer protocol, domain name and, in most browsers, port number. Two resources are considered to be of the same origin if and only if all these values are exactly the same. The policy allows scripts running on webpages originating from the same site to access each other's methods and properties with no restrictions. For webpages on different sites, however, access is denied. This was a real concern for the communication between the server and client developed in this project. Data transfers consists of data in JSON format delivered through AJAX requests. Regular AJAX-calls were prohibited by the browser and failed to work due to the same domain policy. Luckily, the Sencha Touch API has a solution to this problem, so there was no need to create a workaround or find a hack on the web. The solution is called "ScriptTagProxy"[75].

Instead of using the standard AJAX request, ScriptTagProxy injects a <script> tag into the DOM. For example: If the prototype klient in this project wants to load bus-stop data from `http://busDomain.com/stops`, the injected script tag might look something like this:

<script src="http://busDomain.com/stops?callback=someCallback"></script>

[73] `http://en.wikipedia.org/wiki/WebServicesDescriptionLanguage`
[74] `http://www.w3.org/Security/wiki/Same_Origin_Policy`
[75] `http://docs.sencha.com/touch/1-1/#!/api/Ext.data.ScriptTagProxy`

The browser on the client side then makes a request to that url and includes the response as if it was any other type of JavaScript include. Because the client passes a callback in the url above, the busDomain server knows that the client want to be notified when the result comes in and that it should call a certain callback function with the data it sends back as parameters. So long as the remote server is configured to format the response data accordingly, transfers are completed successfully.

### 8.1.4 Optimising Client-side

During testing on the Samsung Galaxy S2, the iPhone 3GS and iPad it was very apparent that the graphics in the user interface of the client was rendered much faster by the iOS-based devices. Presentation of layout, lists, maps and map related graphics like zooming and panning seemed much smoother on these devices. Also, on the Android device, the map flickers during panning, which is really a minor annoyance, but takes away the clean cut look of the application. It is likely that the difference in performance mentioned here is a product of iOS having hardware acceleration, while Android has not. Both iOS and Android devices use the WebKit web-renderer in their default browsers[7]. WebKit already have GPU-acceleration implemented, so the only thing missing is to have native support up and running on the Android devices. The following quote[76] gives an overview of GPU-acceleration on mobile browsers:

> *"Traditionally, web browsers relied entirely on the CPU to render web page content. With capable GPUs becoming an integral part of even the smallest of devices and with rich media such as video and 3D graphics playing an increasingly important role to the web experience, attention has turned on finding ways to make more effective utilization of the underlying hardware to achieve better performance and power savings. There's clear indication that getting the GPU directly involved with compositing the contents of a web page can result in very significant speedups. The largest gains are to be had from eliminating unnecessary (and very slow) copies of large data, especially copies from video memory to system memory. The most obvious candidates for such optimizations are the <video> element and the WebGL canvas, both of which can generate their results in areas of memory that that CPU doesn't have fast access to. Delegating compositing of the page layers to the GPU provides other benefits as well. In most cases, the GPU can achieve far better efficiency than the CPU (both in terms of speed and power draw) in drawing and compositing operations that*

[76] `http://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome`

*involve large numbers of pixels as the hardware is designed specifically for these types of workloads."*

It is a bit curious that Android has not implemented GPU-acceleration yet, but the Android developers have a good reason why. While on other platforms like iOS and Windows Phone 7, minimum requirements for hardware are set in place for devices running their respective operation systems. Not only does this make it much easier to make sure processes run consistently on all devices, but it lets software developers easily implement hardware bound functionality, like GPU-acceleration. To this date, it has been very difficult for Android developers to make a generic solution for this problem. A future release of Android 4.0, nicknamed Ice Cream Sandwich, will supposedly [77] support hardware acceleration and then hopefully make the experience on Android devices similiar to devices based on iOS and Windows Phone 7.

### 8.1.5 Google Maps Woes

Implementing Google Maps in Sencha Touch might seem trivial to implement at first. Only a few lines of code should do the trick, but there are things that have proved difficult. If a map UI object is create without putting it in front and let it render initially, the map does not behave as expected. The map's start position does not get loaded, markers created do not show, loading while panning does not work properly and so on. After some research on the web and looking through APIs, a solution presented itself. Google Maps API provides a way to resize the map and reinitialise it properly. As stated in Google Maps Reference web page [78]:

*"Developers should trigger this event on the map when the div changes size: google.maps.event.trigger(map, 'resize') "*

Putting this line of code in the map object's "activate"-listener solves the problem when the map is rendered initially. Still, there is a problem when an orientation change event fires on the phone, e.g the phone is flipped to the side. The problem is that the resize method needs to be called after the layout has been redone. It is hard to say exactly when the Sencha layout is done rendering after the orientation change since none of the listeners provided gives the correct time of when the map has been fully re-rendered. After extensive debugging and brainstorming, the way around this issue was to put the resize call in a delayed task, i.e. a new thread that runs after a certain amount of time. The delayed task was then run whenever

[77] `http://phandroid.com/2011/10/19/android-4-0-ice-cream-sandwich-has-hardware-acceleration/`

[78] `http://code.google.com/intl/no-NO/apis/maps/documentation/javascript/reference.html#Map`

the application fired a "onWindowResize"-event, i.e. when the application window changes size. A second delay to make sure that the layout has been redone seemed to be a good value during testing.

The Sencha Touch API provides functionality to retrieve and track the users current location easily. It is a simple boolean that is set when initialising the map called 'useCurrentLocation'. This did not work in our device's browsers and we had to create our own method for doing this.

To this date "pinch zoom" in Google Maps is not supported on Android browsers. Implementing custom code for doing this could be time consuming and does not fit the scope of this project. Android users have to make do with the 'plus' (or double tapping) and "minus" buttons on the Google Maps control interface until this issue is fixed in future versions of Android.

## 8.2 Reflections on Creating the Client Prototype

The client application ended up as expected and considered a success by us. Even though the application works more smoothly on iOS-devices, future updates for the Android devices, as discussed earlier, will presumably fix the shortcomings of the Android browser capabilities. As can be seen in the results section, all functionality from Magnus Raaum's application were implemented[13] and the application gives the user the look and feel "illusion" of being native application. The hybrid application strategy worked out as planned and gave us the benefits from using the multiple platform approach we were looking for. We successfully created one application that can be used on both Android and iOS devices without the use of any platform-specific code.

## 8.3 Reflections on Adding a Server and Updating the BusTUC Web-Interface

The implementation of the MultiBRIS server and update of the BusTUC web-interface were considered a success as we see it. The system as a whole went from having an client-application that transferred up to 0,5 MB of data for a bus route query and a query time that was around 20 seconds, to transferring only 5 KB of data and having query times at around 10 seconds. Saving both time and data transfer was imperative for the practical viability of both our MultiBRIS client, and TABuss's Android client.

When looking at the power usage on figure 19 in the result section, that compares power usage when having the business logic on either the client or the server, a surprising property was revealed. One can see that the difference in power usage in a large part comes from the extra data transferred and not from more CPU usage. From what we could find in articles discussing power usage on mobile devices, the CPU cycles and the display are always portrayed as the main battery power consumers. The result in

figure 19 can indicate that data transfers can consume as much power as CPU cycles in some cases.

As shown, there are a lot of benefits from moving much of the business logic to a server. The client saves battery power and it is easier to maintain and update the system. However one should not forget to think about the possible drawbacks from adding a server as part of the solution. Doing this effectively creates another layer where the information has to pass trough to reach the client. Adding another layer creates another point which can potentially fail to work properly.

Another aspect is that the server, when used in a production environment, needs proper infrastructure as a foundation in order to be reliable enough for any client to use it. Though it is not in the scope of this project to look at commercial aspect of the solution, it would be parochial not to point out that infrastructure cost for a server-infrastructure has to considered.

### 8.4 Known bugs

The prototypes have not been thoroughly tested and may therefore have a bugs that were not noticeable during development. The following list depicts those that have been discovered.

#### Client prototype

- The GUI might not rebuild itself properly on orientation change at times (i.e flipping the device to the side).
- The "Trashcan"-button for favourite deletion has been rendered obselete. It is impossible to select a favourite due to instant initiation of searches on tap. The alternative solution of swiping the favourite and clicking the "X"-button works.

#### Server prototype

- On some bus-stops for some routes, the bus departure time is updated with the wrong real-time, this is because BusTUC delivers the wrong bus-stop ID for some bus-stops.

## 9 Future Work

Future work consists of ideas of how to improve or extend the functionality of the MultiBRIS system.

## 9.1 The MultiBRIS Client Applications

This section describes potential improvements specific to the MultiBRIS client application.

### 9.1.1 Back Button support

As of now, the client has no support for device specific buttons, like the back button on Android devices. These buttons runs in default mode. On an android device, the back button makes the user leave the application, which annoys users that expect the back button to work as the "Go-back" functionality in web browsers. This can be implemented with the use of the history states functionality in html5[79].

### 9.1.2 Optimise JavaScript Code to Follow Best Practises

Because of the lacking knowledge of JavaScript at the start of this project, much of the time was spent learning the language and debugging. Little time was allocated to an implementation phase. Therefore, the JavaScript code might not follow what is widely considered best practises and be as optimised as it should be. This, including the use of a MVC-pattern[80], were sacrificed in order to prioritise adding functionality and getting things to work. Future work should include a MVC-pattern for a more structured and tidier code and code optimisations.

### 9.1.3 Multi-Language

Creating language packs and implement an easy way to toggle between different languages in the client application should be an easy task and would open up for foreign users. Buttons, descriptions and results should be translated. Context awareness could be used here. For instance, language in the client application could be set by using the language setting on the mobile device's operating system.

### 9.1.4 Fix XML List of Bus-Stops in Trondheim

The MultiBRIS client application loads bus-stop information from a XML-file. This information is used for both the real-time information functionality and the search textfield auto-complete feature. This file has a few entries that are considered false. The bus-stops might not exist and give errors when posting queries for real-time information. These should be altered

[79] This link should be helpful: `https://github.com/balupton/history.js/wiki/The-State-of-the-HTML5-History-API`

[80] (Sencha Touch supports the use of MVC:
`http://www.sencha.com/learn/a-sencha-touch-mvc-application-with-phonegap/`

or removed accordingly. There are also duplicate names in this list, due to several bus-stops in vicinity of each other (E.g. Studentersamfundet). Future implementations of the MultiBRIS client application should either create a separate list without these duplicates or remove the duplicates by using filters in the code implementation.

### 9.1.5 Improve Euclidian Distance Algorithm

The "Bus-stops near me" functionality uses a simple euclidean distance[81] algorithm to figure out what bus-stops are closest to the user's location. This algorithm does not take the Earth's spheric shape into account. So in order to increase accuracy this algorithm should be improved. This could be done by implementing the same "Inverse formula" from the paper by T.Vincentry as it was implemented on the MultiBRIS server[16].

### 9.1.6 Dynamic Bus-Stop Loading

The current version of the MultiBRIS client application lets the user view the closest bus-stops by the push of a button. The application could be made more intuitive by updating the code to facilitate dynamic loading of bus-stops according to where the user is located or where the user navigates to on the map.

### 9.1.7 GUI Optimalisation for Landscape Mode

The user interface in the application does not work as well in landscape mode (i.e mobile device flipped to the side). The map is very small due to toolbars and menu items. Customisations could be made to free up some space for the map. For instance, it is not essential to have the search bar in this view. The user can easily switch to the "Favourite" view for search functionality. The feature toolbar in bottom of the view could also be removed. The user's current location could be set be automatically updated ("Meg"-button). The clear map functionality might not be necessary for most users ("Fjern"-button). By implementing dynamic bus-stop loading (See previous paragraph) the "Nær meg"-button is also rendered obsolete.

### 9.1.8 GUI Optimised for Desktop Browsers

As explained earlier, the client application is essentially a web application. This means that it already works well with desktop browsers like Apple Safari and Google Chrome. The graphical user interface (GUI) has not been optimised for such desktop browsers, though. The different GUI elements are either too small or they fill the browser window entirely, which is not

[81] `http://en.wikipedia.org/wiki/Euclidean_distance`

optimal for big monitors. Small adjustments should be made to the code base in order to accommodate this.

### 9.1.9 Speech support

Additional support for input and output in the form of speech (Multimodal Interaction[82]) could be added to the client application. This would expand the target audience further to include the visually impaired, elderly and non-natives. The spoken dialog system "Let's Go" (2003) has implemented functionality to make bus route information available for these user groups[15]. For the non-natives, using such a system can even aid in learning the native language[14].

### 9.1.10 Context Aware: Dynamic GUI

When humans communicate, they make use of implicit situational information, context, to increase content and efficiency. To increase the efficiency of the client application it can try to mimic this behaviour by being context-aware[3]. Dynamic GUI based on user's age is one of the things that could be implemented. What this means is to control font size, in addition to what and how much information will be shown. Customising design of the graphical interface and interactive elements, like buttons, could also help achieve this. To keep children interested, one should for instance stick to a minimum amount of text and design the application to have certain shapes and colors to exploit their playful state of mind and draw them in. As we age, perception is lessened, attention is narrowed and memory is limited. Increasing font size and give concise information is therefore preferred for the elderly users.

## 9.2 The MultiBRIS Server

This section describes future work regarding the MultBRIS server.

### 9.2.1 Extensive Testing

Once in a while, during the development of the server, errors occurred that one was not able to replicate and therefore not fix. With many systems working together it is not always easy to pinpoint the exact origin of an error. Therefore it would be beneficial to conduct extensive testing of the server to try to get rid of as many bugs as possible. A proper load test should also be conducted to estimate what hardware specifications is needed in relation to concurrent queries.

[82] `http://en.wikipedia.org/wiki/Multimodal_interaction`

Extensive testing should also be conducted on the client prototype in order to reveal weaknesses and fix them.

### 9.2.2 The Least Transfers Option

A functionality which could be added to the server is to give the user the option to select the route that involves the least transfers. One could even make a weighting system so that the server would suggest a route that involved a transfer only when the total travel time for other routes exceeded a given relative limit.

### 9.2.3 Adding Authentication

At the current stage there is no control on who is allowed to use the server. Anyone that knows the URL can use the service. Adding authentication might be desirable. There are already a query attribute called *key* that can be used to identify the query caller. This attribute currently is only optional. A loose authentication method would be to make this key mandatory, and maintain a list of allowed keys on the server.

### 9.2.4 Compressing the Returned JSON Objects

It could be beneficial to add support for the compression of the returned JSON-objects. This would allow the client side to spend some more CPU cycles in order to save some data transfer. As there are many mobile subscriptions that still have expensive data transfer rates, it could be that some users would prefer to *"pay battery time"* for less data transfered. GZIP is an option in HTTP that provides support for this functionality. GZIP[83] is part of the HTTP 1.1 standard defined by the RFC2615[84]. To our knowledge there is no existing web browser for mobile devices that does not support HTTP 1.1 usage.

### 9.2.5 Caching the Requests

The part of the system that requires most computation is the BusTUC part. One solution to this challenge is to cache of the requests to the MultiBRIS server. This would only come into play in an environment where there are a large number of users. All requests, with responses attached, can be stored for reuse for a given time, typical up to one minute. The reuse time can not be too long, as it could give inaccurate results. Another interesting

[83] `http://en.wikipedia.org/wiki/Gzip`
[84] `http://tools.ietf.org/html/rfc2616`

functionality, in relation to this, would be to use some kind of case-based-reasoning. The server itself can learn how long this reuse period could be extended, for a given route or area, without getting inaccurate results.

### 9.2.6 Filtering Options for the Logging Service

The "logging service" for the MultiBRIS server could be improved by adding filtering options. Typical, useful filters would be logging level, key and date. The level filtering would be handy to filtering out *SEVERE* log entries indicating errors in the system. The *key* would help locate an error that a particular user has, and the date would filter by date. It can also be beneficial to look at how the log is actually stored. Now it uses files with no standard syntax. A standard syntax log could be made in XML[85]. XSLT[86] could be used to transform the XML to HTML content.

### 9.2.7 Intelligent Decisions on Where to Compute

In an paper by Jason Flinn et al. called *Balancing Performance, Energy, and Quality in Pervasive Computing* they portray a system called Spectra[5]. This system dynamical decides whether to perform computation on the server or the client. It makes its decisions by monitoring resource usage both on server and client, and making an "optimal choice" based on given system parameters. This functionality could be implemented for MultiBRIS. For instance if MultiBRIS server was under such heavy load that it would delay query times, the clients calling could be instructed to perform the route calculations and contact underlying services themselves. This would require development on the both the server and the client. The client code would grow substantially, as it has to contain all the business logic needed to perform computations and service call-outs itself.

Another idea from the article by Jason Flinn et al. is allowing the client to learn what is best practise, for instance to save battery power. As one can see in the Result section, battery usage from transferring data can require as much power as CPU cycles. Therefore the client application could monitor power usage for different operations, and learn what tasks to compute by itself and what to ask a server to compute. By doing this, the client could optimise for saving battery on the unique mobile device it is currently running on. The reason why this needs to be learned for every type of device is that battery power used for data communication and CPU usage varies from device to device. Typically, the client could also optimise for lower execution time.

[85] `http://www.w3schools.com/xml/default.asp`
[86] `http://www.w3schools.com/xsl/xsl_intro.asp`

## 9.3 Geographical Expansion

There is no reason why the MultiBRIS system should be restricted to the Trondheim area. Adding support for other cities in Norway, and eventually even the whole country is one of the ultimate goals of FUIROUS.

For instance, public transportation information from the entire country could be maintained in a single system. When a client application starts it could load public transportation information data, from the new system, according to the mobile device's current location.

One of the challenges related to effective expansion is the need for standards. For instance, if the goal was total coverage for all public transportation in Norway, it would be convenient to convince all the bus agencies in Norway, to use the same standards for sharing routes and real-time data. One such standard for real-time data is already in use by Trafikanten AS [87], Norway's largest bus agency. The standard Trafikanten AS uses for distribution of real-time data is called SIRI[88] Service Interface for Real Time Information. SIRI is an XML protocol that allows distributed computers to exchange real-time information about public transport services and vehicles. The protocol is a CEN[89] standard, developed with initial participation by France, Germany (Verband Deutscher Verkehrsunternehmen[90]), Scandinavia, and the UK (UK Real Time Interest Group[91]). Through a JSON-API Trafikanten AS makes the *StopMonitoring (SM)* part of SIRI available for public use. The Stop Monitoring section is as described by the SIRI standard:

> *The Stop Services (Stop Timetable and Stop Monitoring) The Stop Timetable (ST) and Stop Monitoring services (SM) provide stop-centric information about current and forthcoming vehicle arrivals and departures at a nominated stop or Monitoring Point, typically for departures within the next 20-60 minutes for display to the public. The SM service is suited in particular for providing departure boards on all forms of device.*

Because SIRI is a CET standard and already in use by Norway's largest bus agency, SIRI is a good candidate for a national standard for sharing real-time public transport information.

It still remains to find a good standard for sharing route information. However, the biggest challenge when it comes to standards are probably not technical, but rather political and financial.

[87] `http://trafikanten.no/no/verdt-a-vite/trafikanten/Om-selskapet/`
[88] `http://www.kizoom.com/standards/siri/`
[89] `http://www.cen.eu/cen/pages/default.aspx`
[90] `http://www.vdv.de/`
[91] `http://www.rtig.org.uk/`

An approach that avoids the distributed standardisation challenge could be constructed by absorbing the existing transportation agency systems one-by-one. This system would effectually become the mediation layer that creates the standard, seen from an application developer's point-of-view. This would also increase the amount of work needed to expand the system substantially, compared to expanding a system based on standards. An advantage to this approach would be that such a system could establish a position of power in relation to public transport data sharing. It is reasonable to believe that a system that has a standard way to communicate route data for an entire country, would become vastly popular among client developers. By providing the "back-end" to "front-end" mediation layer for the majority of available public transportation client-applications available, one would be in a position of power. This position of power could then be used to encourage the use of standards such as SIRI, among the public transportation agencies.

## 10 Acknowledgments

We would like to thank our supervisors Rune Sætre and Björn Gämback for their guidance. Rune has been a valuable resource, both because of his knowledge of the BussTUC domain, and his general enthusiasm and interest in mobile application development. We would further like to thank Chistoffer Jun Marcussen and Lars Moland Eliassen for their valuable ideas and feedback during the project.

# References


[1] Ken Arnold, James Gosling, and David Holmes. *The Java Programming Language*. Pearson, 4rd edition, 2005.

[2] Adam M. Christ. Bridging the moble app gap. *Noblis. Sigma, Inside the Digital Ecosystem*, 11:27–32, October 2011.

[3] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.

[4] Brian Ferris, Kari Watkins, and Alan Borning. Onebusaway: Location-aware tools for improving public transit usability. *IEEE Pervasive Computing*, 2010.

[5] Jason Flinn, Soyoung Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *In Proceedings of the 22nd International Conference on. Distributed Computing Systems*, pages 217–226, 2002.

[6] George H. Forman and John Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, April 1994.

[7] Edwin A. Hernandez. War of the mobile browsers. *IEEE Pervasive Computing*, 8(1):82–85, 2009.

[8] A. Lenhart, K. Purcell, A. Smith, and K. Zickuhr. Social media & mobile internet use among teens and young adults. *Pew Internet & American Life Project*, 2010.

[9] Christoffer J. Marcussen and Lars M. Eliassen. Tabuss: An intelligent smartphone application. 2011.

[10] Daniel Moraff. Google transit feed specification: A primer. November 2009.

[11] Richard Padley. Html5 - bridging the mobile platform gap: mobile technologies in scholarly communication. *The Journal for the Serials Community*, 24:S32–S39, November 2011.

[12] Bruce W. Perry. *Java Servlet & JSP Cookbook*. O'Reilly Media, 1st edition, December 2003.

[13] Magnus Raaum. An intelligent smartphone application. Master's thesis, NTNU, 2010.

[14] Antoine Raux, Maxine Eskenazi, and This Cmu. Non-native users in the let's go!! spoken dialogue system: Dealing with linguistic mismatch.

[15] Antoine Raux, Brian Langner, Alan W Black, and Maxine Eskenazi. Let's go: Improving spoken dialog systems for the elderly and non-natives. In *in Eurospeech03*, 2003.

[16] T. Vincentry. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, XXIII:88–93, April 1975.