TABuss: An Intelligent Smartphone Application

Christoffer Jun Marcussen, Lars Moland Eliassen

December 20, 2011

Abstract

With the constant increase in smartphone sales, integrated sensors and map navigation has now become available to the average user. This allows for mobile applications to use the user's context to provide more relevant information. An interesting use-case for such applications is a route information systems for buses.

This report covers the improvements made to Magnus Raaum's existing context-aware application for Android, made in 2011. The application uses BusTUC, a reasoning-system for bus routes in Trondheim. By combining user context with BusTUC reasoning and real-time data from AtB, the user-interaction is simplified, compared to a standard information application.

The original Android application had the same basic functionality. We have built on, and extended this functionality, improved the design, and made the application ready for release to the public. Based on feedback from beta-testers, we have reason to believe that the improved application suits the needs of typical bus travellers.

Contents

1 Introduction

This report is the final product by Lars M Eliassen and Christoffer Jun Marcussen for the course TDT 4501, Department of Computer and Information Science, NTNU, 2011.

1.1 Task Description

The task at hand was given by Tore Amble at IDI, NTNU:

> *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 AtB 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 localisation and connections to maps. The project shall take a broad view, and consider all possible advanced concepts, resulting in advanced smart phone applications.*

1.2 Terminology

Explicit terms not covered by the following abbreviations are in this report written in *italic*.

AtB Public transport agency in Trondheim.

HTML HyperText Markup Language, a markup language for formatting web pages.

HTTP Hypertext Transfer Protocol, application protocol for distributed, collaborative, hypermedia information systems.

IDI Department of Computer and Information Science, NTNU.

JSON JavaScript Object Notation, a lightweight data-interchange format.

KML Keyhole Markup Language, file format used to display geographic data in an earth browser.

MultiBRIS Multiple-platform approach to the Ultimate Bus Route Information System, system developed in parallel to TABuss.

PHP PHP is a general-purpose server-side scripting language originally designed for web development to produce dynamic web pages.

SOAP Simple Object Access Protocol, protocol for exchanging information in web services.

SMS Short Message Service, a text messaging service component of phone, web, or mobile communication systems.

TABuss Tore Amble Buss, this project.

TUC The Understanding Computer, a reasoning system developed at IDI. BusTUC is developed for bus route information.

XML Extensible Markup Language, a markup language for sharing structured data.

1.3 Background and Motivation

BusTUC[2] became publicly available to the inhabitants of Trondheim in 1998, based on time tables by the bus company Trondheim Trafikkselskap. Since the commercialisation by LingIT[1] in 2001, approximately one million queries have been asked every year. Now hosted by AtB[2], BusTUC can be accessed through both the web and Short Message Service (SMS).

With the recent progress in smartphone technology, several bus route information applications have become available. Here, we present TABuss, an intelligent application developed to explore new possibilities within the bus route information domain and to utilise more smartphone capabilities. The aim is to close in on the concept of an ultimate intelligent bus route information system.

TABuss is the result of continued development of a previous project[17]. It was initiated by Magnus Raaum, a former M.Sc student at the Department of Computer and Information Science(IDI), NTNU.

[1] http://nyweb.lingit.no/nb/om-lingit/tuc
[2] https://www.atb.no/

1.4 Goals

The goals were based on their estimated time consumption and the time available, both regarding reviewing Raaum's project, and the implementation of new functionality.

1.4.1 Reviewing and Re-implementing Raaum's Application

Raaum's application[17] was not tested thoroughly. User testing is important, to detect *black-box*[3] errors. *White-box*[4] was also performed and it highlighted several flaws leading to frequent crashes when using the application. Testing had to be addressed before adding new functionality, and should also be done before a future release.

1.4.2 Research of Similar Applications

There are several existing applications available using BusTUC and real-time data. Research must be done to gather inspiration, and also to see what the standards are, and what functionality TABuss should contain to be competitive. Existing research performed in the domain of intelligent route information systems should also be reviewed, to guide future development.

1.4.3 Real-time Functionality

Real-time data was already introduced in Raaum's application as a way to improve the BusTUC answers. Real-time data can also be used to: E.g. show the user real-time data for the bus stops closest to the his/her location or any other bus stop. Real-time information itself is not within the domain of artificial intelligence, but it can be used as a valuable resource in intelligent systems.

1.4.4 Artificial Intelligence

The new artificial intelligence functionalities within TABuss' scope, are mainly related to different usages of BusTUC. In Raaum's application, BusTUC was accessed through web communication with a nested syntax. It is a project goal to find alternative ways of handling both the user input and the network communication.

Other interesting features to study are context and context awareness. Possibilities in context awareness, starting with a formal definition of context, are addressed Section 2.10.

[3] http://en.wikipedia.org/wiki/Black-box_testing
[4] http://en.wikipedia.org/wiki/White-box_testing

- Når går bussen fra Samfundet til Torvtaket?(Standard)
- When does the bus departure from Samfundet to Torvtaket?(Standard)
- (Samfundet +n, Prinsen +n) til Torvtaket.(New)
- English translated version not supported.

Figure 1: Standard and new queries. The n represents walking distance (in this case to Samfundet). This representation was discovered to not work properly during testing, as different values for walking distance, did not affect the results.

1.4.5 Improvement of The User Interface

The user interface in Raaum's application was prototypic. The user interface is important for the user experience, e.g: A user is more likely to use applications that are easy to navigate and use. The average user might reject an application after only a few seconds of use, if the user interface is not aesthetically pleasing and intuitive. Therefore, an effort has to been made to design a new user interface that is easy to navigate.

1.4.6 Shifting to the MultiBRIS Server

As mobile devices suffer from restricted hardware compared to stationary devices, performing all the computations on the client-side can affect both performance and battery power. It was therefore decided to shift all the core functionality to an external server, MultiBRIS[5].

2 Existing technologies

In this section we describe the technologies used in TABuss.

2.1 BusTUC

The BusTUC natural language server accepts two forms of queries. One using plain language sentences, referred to as the ”standard syntax”. The second is a merged query format, referred to as the ”new syntax”. Both are shown below.

TABuss uses both syntaxes, where the standard syntax requires a larger amount of natural language input from the user. The user has to enter

Table 1: Bus stop and Real-time ID mapping. New real-time id is assigned when the real-time server restarts.

Bus stop ID	Real-time ID
16000001	111111
16000002	111112
16000003	111113
...	...

sufficient information in order for BusTUC to provide an intelligent text answer. The new syntax allows merged queries. In addition to a plain text answer, BusTUC also returns an object similar to a JSON[5] object. The text answer is not used by TABuss, since all the needed information can be parsed from the returned JSON object.

An example of a returned JSON object is shown in Listing 1.

```
{"transfer":"false" ,"timeset":"false" ,"departures" : [{"
    busstopname":"Studentersamfundet","busstopnumber":16010477,"
    busnumber":5,"time":1139,"duration":11,"destination":"Dragvoll
    "},{"busstopname":"H\o gskoleringen","busstopnumber":16010197,
    "busnumber":5,"time":1141,"duration":11,"destination":"
    Dragvoll"},{"busstopname":"Gl\o shaugen Nord","busstopnumber"
    :16010333,"busnumber":5,"time":1143,"duration":9,"destination"
    :"Dragvoll"},{"busstopname":"G\o shaugen Syd","busstopnumber"
    :16010265,"busnumber":5,"time":1143,"duration":7,"destination"
    :"Dragvoll"}]}
```

Listing 1: BusTUC result with new syntax

2.2 The Real-time System

The real-time system provides information on the actual arrival times of all the buses. Queries and answers are sent and received using a SOAP[6] interface to a server hosted by AtB. The only necessary input parameter is a bus stop's real-time ID. The real-time IDs can be retrieved from the same server as a list that maps each bus stop ID to a real-time ID (see Table 1. AtB updates this list from time to time, and an updated list is necessary in order to query the correct real-time data. An example illustrating bus stop ID to real-time ID mapping, is shown in Table 1.

A query results returns five next bus arrivals for the chosen bus stop. The answer contains route numbers, planned arrival times, actual arrival times and destinations. An example result of a real-time query is shown in

[5]http://www.json.org/

[6]http://www.w3.org/TR/soap/

Listing 2. Descriptions of the key nodes within the JSON object are shown in Table 2.


```
{"total":5,"InfoNodo":[{"nome_Az":"AtB","codAzNodo":"16011265","
    nomeNodo":"Gl\o sh","descrNodo":"1265 (Gl\o shaugen Syd)","
    bitMaskProprieta":"","codeMobile":"1265 (Gl\o shaugen Syd )",
    "coordLon":"10.4087","coordLat":"63.416311"}],"Orari":[{"
    codAzLinea":"5","descrizioneLinea":"5","orario":"06/12/2011
    12:25","orarioSched":"27/11/2011 12:20","statoPrevisione":"
    Prev","capDest":"Dronningens gt.                ","},{"
    codAzLinea":"52","descrizioneLinea":"52","orario":"06/12/2011
     12:31","orarioSched":"27/11/2011 12:24","statoPrevisione":"
    Prev","capDest":"Munkegata - M3 "},{"codAzLinea":"5","
    descrizioneLinea":"5","orario":"06/12/2011 12:37","
    orarioSched":"27/11/2011 12:30","statoPrevisione":"Prev","
    capDest":"Dronningens gt."},{"codAzLinea":"5","
    descrizioneLinea":"5","orario":"06/12/2011 12:48","
    orarioSched":"27/11/2011 12:40","statoPrevisione":"Prev","
    capDest":"Dronningens gt."},{"codAzLinea":"5","
    descrizioneLinea":"5","orario":"06/12/2011 12:58","
    orarioSched":"27/11/2011 12:50","statoPrevisione":"Prev","
    capDest":"Dronningens gt."}]}
```


Listing 2: Real-time query result

Table 2: JSON nodes

Node name	**Description**
nome _ Az	Bus company providing the service
codAzNodo	Bus stop ID
nodeNodo	Shortened bus stop name
descrNodo	Bus stop description
coord(lon/lat)	Latitude/longitude for bus stop
Orari	Contains info on buses passing the stop
codAzLinea	Route number
orario	Bus departure date and time(real-time)
orarioSched	Scheduled departure
capDest	Destination for bus/route nr

2.3 Servers

TABuss has used two servers during the development: *busstjener.idi.ntnu.no* and *furu.idi.ntnu.no*. The server: *furu.idi.ntnu.no* is used in the stand-alone application, while: *busstjener.idi.ntnu.no* provides the MultiBRIS service. In table 3 and table 4 the specifications of the servers are listed.

Table 3: Server information for busstjener.idi.ntnu.no

Attribute	**Value**
CPU	2x 5.2 GHz, VMware shared pool [7]
Memory	4 GB dedicated
OS	Ubuntu 11.04 (GNU/Linux 2.6.38-8-server x86_64)

Table 4: Server information for furu.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

2.4 Android OS

Android[8] is an operating system, originally developed by Google and the Open Handset Alliance, with first release November 5, 2007. Android uses a Linux based kernel containing drivers, with above layers consisting of libraries and the Dalvik Virtual Machine (DVM)[9], frameworks and the top application layer.

As opposed to iOS[10], Android is used by several manufacturers, the top ones being HTC[11], Samsung[12] and Sony Ericsson[13]. Android is licensed under Apache[14], so different manufacturers can adapt their own distributions. This is similar to how Linux has become available in several different versions like: SuSe, RedHat and Ubuntu.

2.5 Android SDK

The Android Software Development Kit (SDK) contains all the necessary classes, packages and files, for developing on the Android platform. The SDK is freeware, and it is available for Windows, Linux and Mac OS. The SDK offers the possibility to target different Android versions, and also provide access methods to device hardware such as the Global Positioning System (GPS), camera and accelerometer. Other features include: media support, database integration and optimised graphics (based on the

[8] http://www.openhandsetalliance.com/android_overview.html
[9] $http://en.wikipedia.org/wiki/Dalvik_(software)$
[10] http://www.apple.com/ios
[11] http://www.htc.com/
[12] http://www.samsung.com/
[13] http://www.sonyericsson.com/cws
[14] http://www.apache.org/

Table 5: Specifications of the devices. Components such as GPS, WIFI and 3G capabilities are not listed, as these are standard on most smartphones released in later years.

Spec	Desire HD	Incredible S
CPU	1 GHz	1 GHz
Android version	2.3.3	2.3.3
Read only memory	1.5 GB	1.1 GB
RAM	768 MB	768 MB
Screen res	480x800	480x800
SD-card slot	yes	yes

OpenGL ES framework[15]). TABuss targets the SDK-version 2.2 or newer.

For Android development, the Android SDK and Java are most commonly used tools. C or C++ code can also be integrated, through the Android Native Developer Kit[16] which can be seen as an add-on to the original SDK.

A central concept in Android development is the *activities*[17]. An *activity* can be described as a main-class, where a view can be attached. An application can have several *activities*, which are managed by an *activity stack*.

2.6 Devices

Two devices were used in the development of TABuss: an HTC Desire HD and and HTC Incredible S. These were chosen because Tore Amble provided the Desire HD, and one of the project members already had the Incredible S.

The two devices have similar specifications (see Table 5). Both devices are powerful, and can handle heavy computations. TABuss does not use heavy graphics or algorithms, which means that devices with poorer specifications can also be used. A absolute requirement is that an SD-card is present and can be used for storage.

[15]http://www.khronos.org/opengles/
[16]http://developer.android.com/sdk/ndk/index.html
[17]http://developer.android.com/reference/android/app/Activity.html

2.7 Maps

TABuss uses Google Maps[18], where a free API key is necessary. As Android is a product of Google, maps are easily integrated using the MapView[19] package. This package provides a wrapper for the Google Maps API, and is used to display maps, and also to add icons on the map[20].

2.8 Application .apk-files

Application Package File(APK) is the file format Android uses for application distribution and installation. It is the output from the compiler, and it holds the source code, the resources and other assets, including the manifest[21] file. The manifest file provides a description of the application name, the activities present and the permissions needed to execute. To install an ".apk", the application has to be developer signed for either debug or release.

2.9 Developing Native Applications vs Web Applications

When developing for mobile platforms, there are several technology decisions to be made. An important one is regarding the choice to develop native applications or web applications. Native development has in later years been the main choice for platforms such as Android and iOS, as earlier versions of HTML did not provide enough framework possibilities[22]. Lately a new option has emerged, with the release of HTML5[23]. Applications written in HTML5 and JavaScript[24], have become the new competitors to native applications. The following sections contain comparisons between native applications and web applications, and the background for the choice to use the native framework in TABuss. The reader is advised to watch talks from the 2011 Google I/O conference[25] for more information on the difference.

[18] http://maps.google.no/

[19] http://code.google.com/intl/no-NO/android/add-ons/google-apis/reference/com/google/android/maps/MapView.html

[20] http://code.google.com/intl/no-NO/android/add-ons/google-apis/reference/com/google/android/maps/Overlay.html

[21] http://developer.android.com/guide/topics/manifest/manifest-intro.html

[22] http://www.w3.org/TR/html5-diff/

[23] http://dev.w3.org/html5/html-author/

[24] http://www.w3schools.com/js/

[25] http://www.youtube.com/watch?v=4f2Zky_ YyyQ

2.9.1 Web Applications

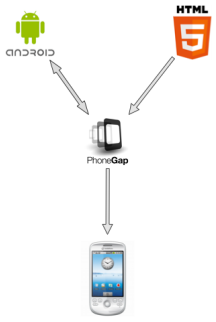

Figure 2: Building a web application

The basic definition of a web application is: "having no ties to a specific operating system or device". Web applications does not rely on any platform-specific API or SDK. They can be designed to resemble native applications regarding the user interface, and they can be deployed on all major platforms. There are many libraries available for development: JQuery[26] and SenchaTouch[27] being two. Performance has also increased with the releases of JavaScript engines such as the Google V8[28].

Web applications can consist of web code only, or they can be a hybrid between web applications and native code. A hybrid application runs the web code, in addition to some parts implemented in native code. This native part can span from simple parts of the user interface, to larger amount of back-end code. What separates a user interface written for web, from a native one, is that the rendering is done by a browser. In native applications, the rendering is done by native graphic components in the device.

The HTML code within web applications can be compiled with a build program. PhoneGap[29] is a build API, which maps JavaScript calls to the correct functionalities found in a native SDK. This results in a build file, which is ready to be installed. These actions are depicted in Figure 1, with Android used as an example.

[26] http://jquery.com/
[27] http://www.sencha.com/products/touch
[28] http://code.google.com/p/v8/
[29] www.phonegap.com

2.9.2 Native Applications

Figure 3: Building a native app

Native applications are developed using platform-specific SDKs. Native applications have (through the underlying operating system) direct access to a device's hardware.

The main disadvantage with native development lies in the meaning of the term. As applications are native, they are not portable to other platforms. An application developed for Android cannot be directly run on iOS, or on any other platform. Another disadvantage affecting the TABuss project, is the access to the Google Maps API. The API on the Android platform is accessed through a wrapper, which has several deprecated functionalities. Figure 2 depicts the compilation of a native application, with Java as selected programming language, and direct access to the SDK.

2.9.3 Comparison

Although web applications can perform many of the same functions as native applications, the end result is not always as satisfying. Web applications (as of today) perform slower than native apps, and large background computing is not supported by build systems (such as PhoneGap[30]). Graphical Processor Unit(GPU)-acceleration is available for iOS, but not for Android versions older than 3.0[31], resulting in poorer performance. Still, it is debatable how many applications that actually need performance at a top level to function properly.

Hardware access has earlier been a problem for web, as the possibilities were limited compared to native. Today, provided libraries give access to components such as GPS, camera and compass, and limitations are less visible. However some are noticeable. An example is the usage of maps. For iOS, rendering results are similar to native, while on Android, performance is slower. "Pinch zoom" is not possible either. Of the tested applications in section 3.2, this was visible in Bartebuss. The map rendering on the iPhone version of Bartebuss, performed much faster than on the Android version. Additionally, problems occurred when turning the screen into/out of landscape mode. This problem has been addressed by MultiBRIS[5].

A fundamental advantage discussed at the Google I/O talks, in favour of native apps, is related to how the web apps can utilise their strength,

[30]http://wiki.phonegap.com/w/page/36752779/PhoneGap Plugins
[31]http://developer.android.com/sdk/android-3.0-highlights.html

namely the paradigm:"code once, deploy everywhere". This is only possible because all technologies based on the web are standardised. The native applications promotor Reto Meier states that: "Standardised technologies will always trail innovation". What this means is that while native developers will have direct access to innovative features, web applications may have to wait until a standard is established. Meier continues with the argument that:"If we look at the recent years' rapid development of new technology such as gyroscopes, multi-touch, tablets and so on, all are based on innovations".

To summarise, both development technologies have advantages and disadvantages. Which one to use depends on the complete context. As mentioned earlier with the argument of Java development being easier than JavaScript, people often chose what they do best. Often, extensive technology research is not possible given strict time limitations, and the learning of a new programming language is not an option. Money also often plays an important part, and the decision can be influenced by the budget. Another interesting aspect is the cost versus profit evaluation. As deployment of an application on multiple platforms may generate more income, it is not an easy decision to make.

Both TABuss, and Raaum's application, are developed as native applications(for Android). As in many other projects, further development of a system often leads to continuing in the same track. Regarding the map problems discovered in Bartebuss, the question is whether or not a web application for this project actually provides full multi-platform functionality, as its performance clearly is poorer than native applications.

Both of the authors are supporters of native apps, which also affected our choice of technology. We see native applications as ideal for research within mobile development. The end goal for research is seldom mass distribution, but proof-of-concept. This can in our opinion be realised by having the best(native) tools available.

MultiBRIS[5] has chosen to investigate and use web technologies. As a result, the FUIROS project may get a comparison of the two through the development process.

2.10 Context Awareness

Context and context awareness have many definitions [15] [21] [20] [4](1997). In the following sections we first describe location-aware computing, before we review some of the existing definitions of context, and check how TABuss fits into these.

2.10.1 Location-aware Computing

Smartphones today are pervasive[32] and personal. This means that they are almost always turned on, and they are customised to each user. Raento, Oulasvirta, Petit and Toivonen(2005) claimed that because of this, smartphones are well suited for context-aware applications[18].

Hazas,Scott and Krumm(2002) stated that: *context awareness is at the core of location-aware computing[10].* Location-aware systems use the user's location in functionalities, through a location-sensing technology. A location-sensing technology is not restricted to GPS or WiFi, but also includes Radio frequency identification(RFID)[33]. RFID tags have a short transmission range compared to GPS and WiFi. An example of RFID use in a context aware application is to place tags in door entries, and track passing people.

A domain of context awareness is in tourist guide applications. Wang, Sørensen, Brede, Servold and Gimre, in 2005, developed the context aware *Nidaros Framework*[27]. Using this framework they implemented a tour guide for the Nidaros Cathedral[34], which uses the user's location to display zones and nearby objects.

Because of the arguments stated by Raento, Oulasvirta, Petit and Toivonen, we claim that the domain of bus route information systems is well suited for context awareness. Such systems should be able to monitor the user's behaviour through sensor input, and use this data to provide route suggestions or other information.

2.10.2 Definitions

Pascoe (1998) defined context to be "a subset of physical and conceptual states of interest to a particular entity"[15], where the importance of the involved states had to be determined. An example of an entity could be a person.

Schilit and Theimer(1994) defined context by three factors: location, descriptions of people in the immediate surroundings and objects with the changes these objects go through[21]. This is a broad definition, with essential criteria needed to be fulfilled, in order to be classified as context.

Ryan et al. (1997) defined context as the user's location, environment, identity and time[20]. Context is generalised by including a number of physical and logical attributes, assumed to affect the user's environment. The definition has similarities to Schilit and Theimer's 1994 proposal, in terms of important factors. However, Ryan et al.(1997) introduces time as an additional factor.

[32]http://en.wikipedia.org/wiki/Ubiquitous_computing
[33]http://no.wikipedia.org/wiki/Radio_Frequency_Identification
[34]http://en.wikipedia.org/wiki/Nidaros_Cathedral

Dey (1997) defined context in a more general formulation, and not by enumerating a list of factors needed to be matched, in order to be called context: "context is any information that can be used to characterise the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and application themselves"[4]. This formulation simplifies declaring functionalities and theories as context. If any part of information can describe or help the user at a given time, it can be called context.

Our project uses *context* and *context awareness* in two different ways. The former context uses the user's location, while the latter context introduces time and destination as additional factors. Schilit and Theimer's(1994) and Ryan et al.'s(1997) definitions, would consider the limited and changing factors not to be adaptable to our purpose, as this would lead to forcing our factors into their definitions.

Pascoe's(1998) definition cannot be used, because of the lack of modularity. If adding additional factors to determine context, their importance would have to be determined: this is difficult, as situations change(what is rated as an important factor in one setting, might be less important in another).

Dey's(1997) definition is the one best suited for our project. This is because of its generality, which defines a factor to be a part of the context, as long as it concerns the user. It also allows for context to be either implicit or explicit. This means that context information is both provided by the user and automatically detected by the system. Dey also defines "context awareness" as the system's response to changed context. It does not determine whether the system should initiate an action automatically or not(in our project, this fits). In a practical example from TABuss: the system should not automatically start downloading the real-time data for one or more stops, only because the user has changed his/her location. But in an intelligent application, it is still an advantage to have the option to do so.

As mentioned earlier, context and context awareness are used in two different ways. Context is used during the general location tracking of the user, and subsequent loading of nearby bus stops. This happens dynamically as the user moves and can be seen on the map while moving: clickable bus stop icons are added or removed as the user changes his/her location. This is an example of implicit context: the user does not provide the location information manually(location is automatically detected and tracked by the system). However, a location technology such as GPS, WiFi or 3G must be enabled.

The second way, which uses more factors(e.g. time and day), is described in Section 3.6.4.

3 Method

In this section we first identify solutions similar to TABuss, found in Trondheim and in other parts of the world, before we describe the development process.

At the beginning of the project, MultiBRIS[5] planned the implementation of a server hosting the application logic, including the communication with BusTUC[2] and AtB's real-time system. The development sections first describe the implementation of TABuss without thinking about the server functionalities. Then, the shifting to the server and necessary adjustments are described.

3.1 Raaum's BusTUC Android Application

BusTUC Android Application is the application that was created by Magnus Raaum[17]. It displays a map with user and bus stop locations, and an input field for entering the desired destinations. The system uses the closest bus stops to a user's location to search for possible routes, by sending queries to BusTUC. Received departure suggestions are then updated with real-time departure times, before they are shown to the user as plain text suggestions.

3.2 Existing Solutions in Trondheim

The following sections describe existing applications, developed for getting info about bus transportation in Trondheim. Table 3.2 gives a summarised

Table 6: Functionality in the tested applications.

Application	Bartebuss	Alf's Bybuss	Bussdroid	Busstider
Bus oracle	Yes	Yes	Yes	Yes
Map	Yes	Yes	No	Yes
Favourites	Yes	No	No	No
History	Yes	Yes	Yes	No
Real-time	Yes	Yes	Yes	No

comparison of the tested applications. Below a brief description of each criteria is given. The following sections describe all the tested applications.

BusTUC Whether the tested application uses BusTUC as a functionality

Map Whether the tested application integrate maps

Favourites Favourites functionality allows a user to store queries or specific bus stops for later use. A typical usage is to store a query with an additional tag

History History defines earlier searched queries or selected bus stops, stored in the internal or the external storage

Real-time Whether the tested applications provide real-time data.

3.2.1 Bartebuss

Bartebuss[35] is developed in HTML5 by Rune M. Andersen, and uses the *BusBuddy* API[36]. It has rich functionality, with options to store favourites, find near-by bus stops, search for specific bus stops, use BusTUC and show maps. Due to the use of HTML5, the map is not as responsive as in a native application. The user interface on the other hand is intuitive and easy to navigate but it might provide too many choices to the user.

Figure 4: Bartebuss

There are 5 ways to use the application to find bus departures. The first is through ”favourites”, where several bus stops can be stored(giving quick access). When a favourite is selected, real-time data for the bus stop is displayed. The second way to use the application is an option called ”near me”, where the closest bus stops to a user’s location are listed. Items can be selected the same way as in favourites. The third option is ”search”. Here, the user can search for a specific bus stop, and view real-time data. The fourth option is BusTUC, where a text field is used for query input. The last option is to select a bus stop icon from the map, which triggers a rewrite of real-time data. The map part was lagging during testing on the Android platform, and hardly usable at all. Two other

[35]http://bartebuss.no/om
[36]http://busbuddy.norrs.no/

problems are the lack of pinch zooming and poor handling of landscape/-portrait changes(as resizing causes problems). When tilted to landscape mode, the zoom controls disappear.

3.2.2 Alf's ByBuss

Alf's ByBuss [37] is a native Android application, developed by Alf Simen Sørensen. It also uses the *BusBuddy* API. It has a simple user interface, and provides BusTUC functionality, with the option to include the user's location as the departure parameter. The application has a text field for entering queries, and some additional functionalities in the menu (activated by the menu button). The main component of the user interface is a map showing clickable bus stop icons. The user can select the bus stop icons, with corresponding bus stop names, as departure and destination, and view real-time data. Menu options include: "Use my existing location"(which inserts the user's current location into the text field as departure input), "reverse search"(which switches the departure and destination stops), and a link to the online bus schedules. *Alf's ByBuss* appears more responsive than *Bartebuss* during map navigation, but the user interface is not as polished.

3.2.3 Bussdroid

Bussdroid [38] is a native Android application by Ken Børge Viktil. Unlike the two previously tested applications, it does not provide a map. The functionality consists of real-time data for bus stops, a BusTUC query text field and the possibility to store queries as favourites. The application is responsive, and has a clean and intuitive user interface.

3.2.4 Busstider

Busstider [39] is a native Android application by Martin M. Syvertsen. Functionality is limited to BusTUC, and a map displaying bus stop icons. Real-time data is not available. There are two ways to use the application: 1) Asking BusTUC with a natural language query, 2) selecting departure and destination by clicking icons on the map. It has a simple and intuitive user interface based on tabs, and is fairly responsive.

[37] http://bybuss.alfsimen.com
[38] market.android.com/details?id=com.ken.bussdroid
[39] http://www.a2bsoft.net/projects/busstider

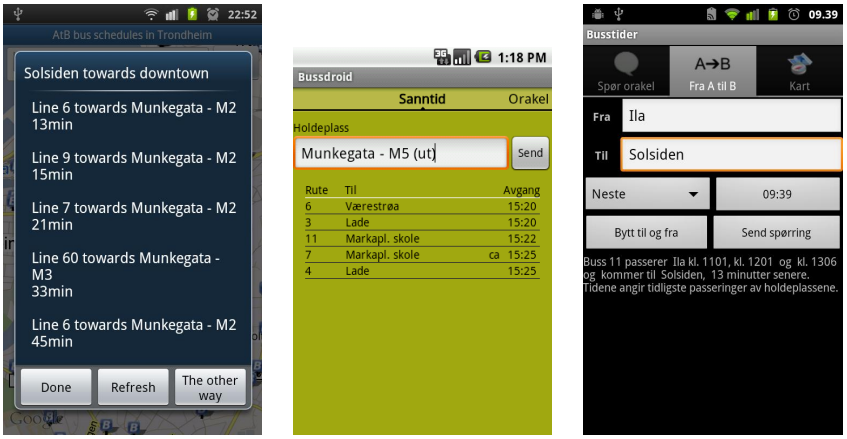

Figure 5: From left to right: (1)Alf's ByBuss, (2)Bussdroid, (3)Busstider

3.2.5 BusBuddy API

BusBuddy[40] is an API developed by Roy Sindre Norangshol, aimed at the retrieval of real-time data. It is a hobby project with the goal to minimise the overhead of SOAP messages downloaded to the mobile devices. Both *Bartebuss* and *Alf's ByBuss*, use this API. All applications developed with the *BusBuddy* API are available through GIT Hub[41], as *BusBuddy*'s policy is that development projects should be open source.

In the early stages of the project using this API was considered . However, the disadvantages lead to discarding of the idea. As it is a hobby project, retrieving a license key was not easy. In addition, *BusBuddy*'s username and password for access to AtB's real-time system is not permanent. AtB can retract this at any time. The process of retrieving new keys, would have to go through the *BusBuddy* crew. By using the username and password provided by LingIT, this "middle man" operation is avoided. It is also likely that AtB would renew the username and password quicker for LingIT, than for a private operator.

3.2.6 Summary of Existing Solutions in Trondheim

All the tested applications for bus route information in Trondheim are already downloaded by several users, indicating that they have attractive functionalities. For our project it was important to investigate what has to be done to move the concept of a bus route information application to a new level. And also to compare the different levels of artificial intelligence in the apps.

[40] http://api.busbuddy.no/
[41] https://github.com/

Especially *Alf's ByBuss* and *Bartebuss* are close to our goals: they use BusTUC, maps and real-time functionality. Both *Alf's Bybuss* and *Bartebuss* integrate natural language through BusTUC, but aside from this, they do not have any other functionalities involving artificial intelligence. Their functionalities are based only on user input and menu navigations. The main topic of our project is reducing user input. In Raaum's application, route suggestions were calculated based on the closest stops to the user's location, and were automatically updated with real-time departure times. This is the core of the development of TABuss. Though the display of real-time data when a user presses a bus stop icon is an important functionality, we have to keep in mind the artificial intelligence aspects. This can separate our project from *Alf's ByBuss* and *Bartebuss*, and give us an upper hand regarding market potential.

3.3 Extended Research

Even though there are several applications available in Trondheim for bus route information, an expanded view including other cities and research performed is more informing. As discussed in Section 3.2, aside from the general use of BusTUC, no functionalities in the tested applications can contribute to moving TABuss forward in the field of artificial intelligence. This section describes research areas within bus route information systems, with the main focus on intelligence through natural language, and the use of real-time data.

3.3.1 Natural Language Applications

In intelligent route information systems, natural language has been addressed by different reseach papers[19][25][24][23].

Raux et al.(2003) developed a system called *Let's go*, which uses speech through phone calls as input, for returning route information[19]. The system was developed for "elderly and non-native English speakers", and provided information for the city of Pittsburgh. Speech was recognised by comparison and retrieval of the closest match. Emphasis was on creation of a grammar model for spoken language, and to include an overall generality regarding different structuring of sentences with the same meaning. Through their work, Raux et al. identified several challenges with speech processing and route information. The main challenge was that different users structured the same phrases differently, when referring to bus stops or places. Hypothetically, assuming BusTUC has a similar up to date speech system (*TeleBuster*[42] is not in use), this system would have to be able to infer correct mappings from spoken text, and also be able to extract content from text spoken with different dialects. As Trondheim is a city with

[42]http://www.idi.ntnu.no/ tagore/telebuster/

inhabitants from different areas of Norway, this could become a complex operation.

Turunen et al. (2007)[24] (2006)[23] proposed a similar solution, *TravelMan*, developed for the city Tampere in Finland. *TravelMan* is an updated version of *StopMan*[25](2006), which provided route planning. Input consisted of locations or addresses, provided to the system by text or speech. The user could also set personal preferences, such as exclusion of transportation options, as *TravelMan* in addition to bus transportation, covered metro and tram. Of other features, a guidance functionality for visually impaired users was implemented, to provide what was referred to as an "unbroken trip chain". An unbroken trip chain was a successful trip, completed with full system guidance.

An interesting feature in *TravelMan* related to our project, is the use of context and user location. The real-time guidance relies on location information, which also could be used to infer departure addresses. A video of TravelMan in use is available at YouTube [43]

3.3.2 Real-time Bus Information

Early research focused on finding an optimal route. The Traveling Salesperson Problem [3] is transferrable to route information, and several algorithms have been proposed. An example is Robert J. Szczerba et al.'s (2000) paper, which described an adaptation of the A*[9] algorithm, for finding the optimal route in real-time[22]. Assuming bus route companies incorporate a similar feature when planning routes, an algorithm including traffic information could give a real-time estimate of where the bus is at a given time.

Maclean and Dailey's *MyBus*(2000) predicts real-time arrival of buses by using historical data, the bus schedule and an prediction algorithm[13]. This algorithm, based on Kalman filters [11], produced estimates, where traffic and passenger information is used as noise, affecting the original schedule.

Maclean and Dailey also published an article on a *Mobile MyBus*(2001), which used communication through WAP[14]. This system gave users real-time information based on received input. The input consisted of destination and a route number, sent by a URL request, and forwarded to the *MyBus* server. Both were provided as digits as devices at that time did not have any input mechanism similar to a computer keyboard. In our project, this functionality can be compared to retrieving a real-time ID for a bus stop, before sending a SOAP-request to fetch real-time data.

[43]http://www.youtube.com/watch?v=bPVAQtHtC3s

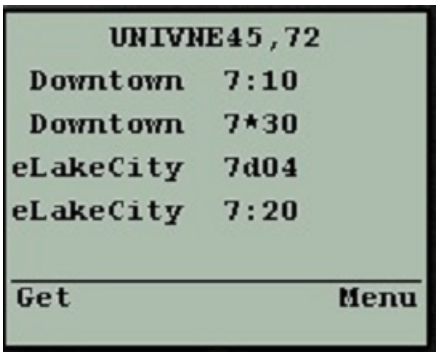

Figure 6: MyBus WAP

Today, buses have GPS-trackers on-board, transmitting location information to a server. Real-time data is updated at specific time intervals and is fairly accurate. The real-time service was introduced in Trondheim by AtB. A similar service is provided in Oslo[44].

Ferris, Watkins and Borning's *OneBusAway*(2010) [45] focuses on context awareness in addition to the use of real-time data[6]. *OneBusAway* uses the location of the user to automatically display the closest bus stops on a map, similar to TABuss. Context aware functionality proved to be successful through user testing, where 93 per cent of the existing users of *OneBusAway* reported that more concise information was provided.

What differs *OneBusAway* from TABuss, is that TABuss' main query functionality only needs the users destination as input. *OneBusAway* needs to know both the departure stop and which route to select in order to reach planned destination.

3.3.3 Summary of Extended Research

Both *TravelMan* and *Let's Go* utilise speech recognition to provide route information.*TravelMan* extended the concept even further by introducing real-time guidance based on user location, and had features worth researching for future functionality. It was shown through Turunen et al.'s work that natural language has potential for mobile development and route information applications.

BusTUC has since its release become a popular choice in Trondheim. As users have become familiar with the usage of natural language, a logical next step could be to introduce speech. iPhone 4S has implemented a system called SIRI[46], which also contributes to familiarising people with natural language.

Research on the usage of real-time information has shown progress; from using estimated real-time, to actual real-time data. Maclean and Daley's *MyBus* WAP version was used over 9000 times, in a time period between its release in September 2000-January 2001, giving an indication that real-time route information on mobile devices is a promising field.

[44] http://trafikanten.no/

[45] www.onebusaway.org

[46] http://www.apple.com/iphone/features/siri.html

3.4 Testing

In this section we identify several existing problems that will be fixed in this project, or in the extension projects in the spring 2012.

3.4.1 Testing Raaum's Application

Raaum's original source code compiled and ran "out of the box", and only required a switch of the API-key, to get the Google Maps integration working. It crashes occasionally, when not reaching the BusTUC server, caused by exceptions that are not caught in the source code. To test the application, several queries were executed, and success rates and response times were monitored. The response times varied between 20 and 40 seconds, which was much slower than the applications reviewed in section **??**. It is worth mentioning that the 20-40 seconds monitored for a run includes computations performed with multiple bus stops. A query with one bus stop, took 5-10 seconds to perform on average.

Table 7 displays the time needed to perform five queries, with origin at Gløshaugen and destination at Ila.

Table 7: Tested queries.

Run	1	2	3	4	5
Time BusTUC	18 sek	17 sek	19 sek	16 sek	19 sek
Time Real-Time	5 sek	7 sek	7 sek	6 sek	7 sek
Time Total	23 sek	24 sek	26 sek	22 sek	26 sek

The results show that the queries to BusTUC were the most time consuming, while the real-time queries were much quicker.

Slow query runs were caused by the BusTUC server, located at Gløshaugen. The other applications tested, used servers hosted by Amazon[47]. The Amazon servers are faster as a result of threading and the use of sockets, performed by a Python script. The server at Gløshaugen uses a slower approach with a PHP-script that runs in the background. This script reads queries from files, and writes queries and results to files. The Prolog side of BusTUC checks these files at given intervals, and processes the queries.

The two different approaches give two different responses: to use the BusTUC hosted by Amazon, only the standard BusTUC syntax(defined in section 2.1) can be used, and the results are returned as text. The BusTUC hosted at Gløshaugen returns a parseable JSON object, in addition to the text, if the new BusTUC syntax is used. In Raaum's application, the latter format was required for the real-time updates of route suggestions. An

[47] http://aws.amazon.com/ec2/

uncaught exception was thrown if the returned JSON objects contained errors. As a remark, the returned JSON objects from BusTUC does not follow correct JSON syntax. The JSON objects contain "
" html-tags, which have to be removed for a parser to be able to recognise the structure.

To benefit from using the faster Amazon servers, some adjustments have to be made. The servers do not include JSON objects in their results, and a direct swap is not an option as important information is lost when using only the text answers. This JSON information includes bus stop IDs and transfer details. A switch to the Amazon servers would also complicate debugging. Debugging the server at NTNU can be done *white box*, as root access is available. It is unlikely that we would get access to the Amazon servers.

Figure 7: Bus stop icons

Raaum's application uses an XML-file containing: bus stop locations, bus stop names and bus stop IDs, which does not contain sufficient information: information for several bus stops are missing. Also, the list also only contains one bus stop description per "bus stop group". A bus stop group consists of one or more bus stops with the same bus stop name. This is a problem in the extraction of real-time data, as a unique bus stop ID is necessary for each bus stop in a bus stop group. Figure 7 illustrates this problem: only one icon is displayed for each stop group.

Raaum's application has two sorting errors: one when the total travel time is equal for more than one route suggestion returned by BusTUC, and one when the distances are equal to two or more bus stops from the user's location. The error source is the use of HashMaps[48], with total times or distances from the user's location to bus stops as keys. Keys in HashMaps are unique, because multiple equal keys overwrite the previous mapped value. In the total travel time issue, this leads to an exception as one or more route suggestions are overridden. In the distance issue,the error leads to a wrong display of bus stop icons on the map.

Raaum's application occasionally crashes during parsing of real-time

[48]http://docs.oracle.com/javase/1.4.2/docs/api/java/util/HashMap.html

data. This happens because the wrong node is extracted for some numbers. The extracted route numbers sometimes contain the letter "N", which in turn causes an exception when parsed as an integer.

Queries with the new BusTUC syntax were meant to incorporate walking time to a bus stop in minutes. After testing, this was discovered to be ignored in BusTUC's reasoning. Time offsets may assure that unrealistic route suggestions are dropped. But on the other hand, real-time data is only available for bus departures in the "near future", and later real-time departure times are equal to the scheduled time. Syntacticly, it is a challenge to add text in the nested query syntax. A working syntax was: *(Samfundet +2) fra Samfundet til Tiller etter n*. This was reasoned by BusTUC as if we wanted a bus departure after *n*.

Another issue is the handling of transfers, where two or more bus travels are needed to reach a destination. Raaum's application often suggests routes where the transfer bus departs before the first bus has arrived. This happens when the real-time data replaces the scheduled departure times returned from BusTUC, and no further sanity checks are performed. Although this does not cause an exception, the returned answers are unintelligent.

The application crashes every time *Lade* or *Ranheim* is entered as input, while *Lademoen* and *Ranheim Stasjon* works. Testing with the BusTUC web-end and standard BusTUC syntax, works for all four. When the standard BusTUC syntax is used, BusTUC provides the correct bus stops. For Lade, this results in *Lade allé*. No reasoning happens when using the new BusTUC syntax, except when a transfer is involved. BusTUC then maps the location to the bus stop correctly. Two examples are the queries: *(Gløshaugen nord +2) til Lade*, and *(Torget +2) til Lade*. The first query is successful, as a transfer is necessary to get to Lade. The second query leads to an exception.

The list below summarises the errors and exceptions found and corrected during testing and debugging. There was no handling of these exceptions in Raaum's source code, but it was important for our future versions to introduce the necessary exception handling.

- No handling of wrong or empty input
- No handling of busy server
- No handling of empty query results
- Insufficient bus stop information
- Wrong sorting of route suggestions
- Wrong handling of routes involving transfers
- Wrong parsing of real-time data

- Wrong display of a user's location
- No handling of missing internet location

3.4.2 Adjustments Made to Raaum's Application

It was necessary to modularise the existing source code because Raaum's code structure relied on to many dependencies, making any modifications difficult. Exception handling has also been addressed, for future development and for user testing. When Raaum's application was tested on Android devices(unplugged from a development machine), no detailed exception description was provided to the user. The only feedback was: "The application has closed unexpectedly". This needed to be improved because the average user is not capable of accessing exception descriptions through a debugger.

The HashMap issues were solved by re-implementing the sorting algorithms, and choosing a more object-oriented solution. HashMaps are fast when performing look-ups, but their usage did not fit this project. An object-oriented solution also aided the separation of code into multiple classes and activities.

The HTML answers returned by BusTUC contained both JSON objects and text. The text answers were not used by Raaum's application, and should not be a part of the result returned from BusTUC. This is a BusTUC related issue, and not within TABuss' scope.

The transfer time issues were fixed by adding a validation after a route suggestion was received. The solution was simply to calculate the user's arrival time at the second bus stop, and compare with the real-time departure time.

```
depTime1 = Departure time from the first bus stop
depTime2 = Departure time from the second bus stop
walkingTime = Estimated two minutes of walking time
travelTime = Travel time from the first bus stop, to the second
    bus stop
if (depTime1 + travelTime + walkingTime < depTime2)
  discard suggestion, and calculate a new route with updated
      arrival time at the second stop

 return suggestion
```

Listing 3: Real-time query result handling with logical soundness checks

An example BusTUC query for a new route, is: *(Sentrum+2) fra Sentrum til Ilsvika etter 1900)*.

3.4.3 Testing BusTUC and The Real-time System

The numbering scheme for bus stop IDs causes some problems: if there are two or more stops in a group, they can be separated by the fourth last digit in their IDs. This digit identifies whether passing buses are headed towards or away from the city centre. If there are two stops, buses heading towards the city centre pass the bus stop with 1 as the fourth last digit of its bus stop ID. Buses heading away from the city centre then pass the bus stop with 0 as the fourth last digit of its bus stop ID. An example is the stop group at *Ila*:

- Buses heading towards the city centre: 16011192
- Buses heading away from the city centre: 16010192

Problems occur for stop groups which only have one stop. Rune Andersen[49]received emails from AtB, where it was explained that stop groups with one stop were assigned a bus stop ID with 0 as the fourth last digit. If we use *Gudes gate* as an example, figure 8 implies that passing buses are headed away from the city centre("fra byen"). However, a look-up in with AtB's route schedules shows that all buses pass Gudes Gate on a route heading towards the city centre. A BusTUC query from *sentrum* to Gudes gate suggests a route from sentrum to the end stop *Asbjørnsens gate*, and then down to Gudes gate. Instead, the user could get off the bus on its way to the end stop, and walk(100 m) to Gudes gate.

Another problem with BusTUC appears for queries with route 63. Then, the same bus stop ID is returned regardless of the direction of the passing buses. The tested queries were: *Ila* to *Dragvoll* and Ila to *Ilsvika*. Ila to *Buenget*, which is in the same direction as Ilsvika, returned the correct ID, as route 5 was returned instead of 63. Clearly, there are some inconsistencies, which seem to only affect certain stops and routes, but which can become problems when implementing new functionality. The project's supervisor(Rune) thought this occured as certain routes are "circle" routes, where buses only ever pass one stop in a stop group. However, for route 63, buses pass more than one of the stops in several of the stop groups. Still, only one bus stop ID is stored in BusTUC's knowledge base. This problem further affects Raaum's application's real-time updates of the departure times. Route suggestions are possibly updated with real-time departure times for buses travelling in the wrong direction: the two queries: *(Høgskoleringen +2) til Ilsvika* and *(Høgskoleringen +2) til Asbjørnsens gate*, return the same bus stop ID for route 63. Updating with real-time data returns equal departure times.

[49]http://www.ntnu.no/ansatte/rune.andersen

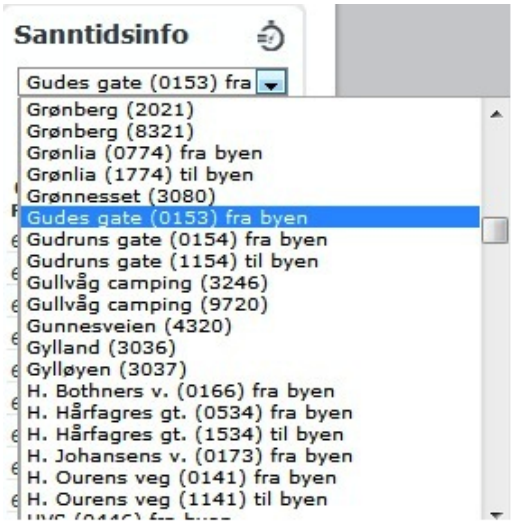

Figure 8: From atb.no: Gudes gate, with wrong direction given

3.5 Finalising Raaum's Application

Raaum's application has several errors (see Section3.4.1), and finalising his application was prioritised before the implementation of new functionalities.

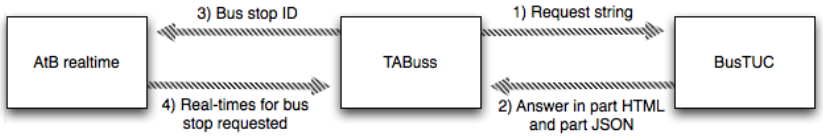

Figure 9: Query functionality

3.5.1 Location

The user's location is retrieved through a position technology such as WiFi, 3G or GPS. The accuracy depends on the choice, as WiFi provides higher accuracy than 3G. The reason for this, is partially controversial[50]: Google scanned and stored MAC-addresses in the process of developing Google Street View.

Raaum's application has problems displaying the location of the user. The marker used to display the user's location moved when the user zoomed. There is also no indication of how accurate a location fix is. This is solved

[50]http://www.wired.com/threatlevel/2010/05/google-street-view-cams/

by the implementation of an Android built-in overlay[51], where a circle is drawn to display the user's location. The circle's diameter is determined by the accuracy of the location fix. Raaum's application also does not give the user any feedback if a location fix is lost.

When TABuss starts, it retrieves a location fix and does not proceed until one is present. How long this takes is dependent on location technology, e.g Edge is likely to use more time than WiFi. If the location fix is lost, a warning is given if the user tries to use functionalities dependent on his/her location.

3.5.2 Map with Bus Stops

The displayed map is re-implemented to allow touch events, triggered by pressing bus stop icons. The map is changed to use an "ItemizedOverlay" [52], which the bus stop icons are added to. The XML-file containing bus stop information is also replaced, and TABuss now uses two bus stop lists. One contains information on all bus stops, and one only contains information on one bus stop per bus stop group. The latter is used in the BusTUC query functionality, where only a bus stop's name is required. If we use the list with information on all bus stops, duplicate bus stop names are included in queries. This happens because bus stops within a bus stop group often are located close to each other.

An example of the content in the bus stop lists is shown in table 8. Each list element contains: The bus stop's ID, the bus stop's name and the bus stop's coordinates.

Table 8: Bus stop list

ID	Name	Latitude,longitude
16538544	Øie skole	10.254138,63.32273
16011292	Marcus Thranes vei	10.367198,63.35539
16011374	Ranheim idrettsplass	10.521225,63.42812
16010258	Anders Buens gate	10.429856,63.43846
...	...	...

3.5.3 User Feedback

An important part of the user experience is to get feedback when errors occur, caused by either faulty user input or a system failure. A system failure in Raaum's application involves errors from BusTUC, the real-time

[51] http://code.google.com/intl/no-NO/android/add-ons/google-apis/reference/com/google/android/maps/MyLocationOverlay.html

[52] http://code.google.com/intl/no-NO/android/add-ons/google-apis/reference/com/google/android/maps/ItemizedOverlay.html

system and the application itself. In TABuss, feedback is returned to the user both when actions are performed and when exceptions are caught:

- Progress bars for start-up operations
- Progress bars for query runs
- Error message for missing internet connection
- Error message for missing location fix
- Error message for missing mounted SD-card
- Error message for no result found for query
- Error message for empty input in text fields

3.6 TABuss

The following sections provide an overview of the development of TABuss.

3.6.1 Development Framework

The development of TABuss has followed Scrum[53]'s guidelines for iterative development.

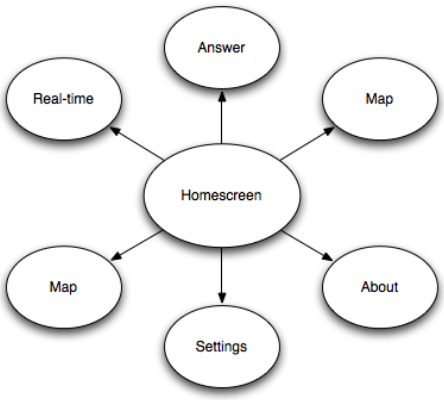

Figure 10: Activity overview

[53] www.scrum.org

3.6.2 Architecture

The development part focuses on making usage of the application as easy as possible. By encouraging thumb-navigation, the map is reduced to an extra feature. Raaum's application runs only one activity, which also includes the map. This means map info is downloaded every time the application runs. To avoid this, TABuss is divided into several activities, where the top activity defines a home screen. Menu and button presses start other activities, and the users can choose for themselves whether or not to use the map.

Figure 11: Screenshots of Raaum's application.

3.6.3 User Interface

The user interface in Raaum's application consists only of the screen in Figure 3.6.2. Queries are sent and answers displayed in the top section of the screen, while the bottom section displays the map.

One of the most important principles when designing a user interface, is simple and intuitive usage. Extended functionalities require more screens, and has to be solved carefully to achieve user friendliness.

According to [8], one of the challenges in mobile computing is the small displays. Even though this article was written in 1994 and the size and resolution of mobile devices have increased since then, it remains a challenge. While a laptop-computer has a 15 inch screen, the minimum requirement for an Android device is 2.5 inches and QVGA resolution (240 $\times$ 320 pixels)[54]. To cope with the small screen sizes, the amount of elements in the user interface is reduced to a minimum.

> *The acronym KISS (Keep It Simple, Stupid) applies well to interface design. A simple, effective interface should be designed with the users' needs taking first priority.* [16]

[54] http://source.android.com/compatibility/index.html

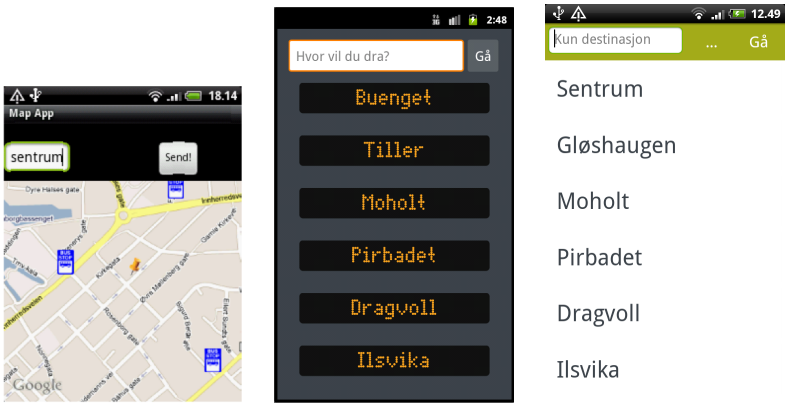

Figure 12: a) Raaum's home screen, b) First draft, c) Final draft

The user interface is designed following the "KISS" and the "Less is more" principles. We want to keep it as basic as possible, while still being aesthetically pleasing. Icons are not used as we want focus on the displayed text.

We chose to use the colours from the AtB website[55], as these colours are associated with buses by the people in Trondheim. An early version of the user interface had buttons resembling the LCD signs found on the front of buses, but as the user interface in other features of the application did not have a similar look, a more simplistic approach was chosen in the end. Contrasting colours are used to make the text visible under various lighting conditions.

The route suggestions in Raaum's application are text based. This is not a very good solution for handheld devices with small screens,and is redesigned to show only the most important information, where intuitive layout replaces unnecessary text.

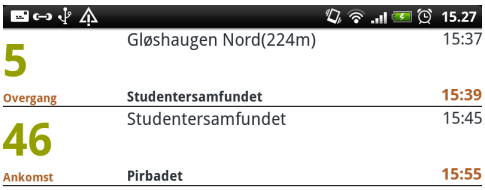

Figure 13: The answer screen.

[55] www.atb.no

3.6.4 Context Awareness

Context is mainly extracted from the user's location. The device's location listener automatically loads the bus stop objects(from the bus stop list) closest to the user's location, when a location change has been triggered. Real-time data for these can be accessed from the map, or through a list available in the menu. The closest bus stops also play an important part in the main query functionality, where the user's location determines which bus stops are included as departure stops.

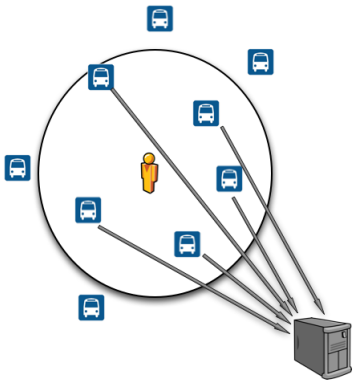

Figure 14: Context retrieval

To distinguish TABuss from the many existing solutions, extra functionality is added by giving the user the option to let the application guess where he/she is going. A simplified version of case-based reasoning [1] is implemented, by logging each made query as a case. These data are stored locally in a database, where each case consists of: The departing area, time of day, day of week and destination. Departure areas are squares of 500 $\times$ 500 metres, with defined area codes stored in a separate table. Whenever a new log item is created, a new area is created if the origin location is not covered by an existing area.

To retrieve relevant cases, queries with similar origin and time are fetched from the database. Similarity is implied by identical areas and somewhat similar time of day. For now, +/- 2 hours is used. The retrieved cases are rated by the euclidean distance between each case, and the current time and weekday. The best matching destination is then presented to the user. +/- for hours is used as finding a direct match is difficult. We want to also include delayed bus departures, and bus departures from within a time period.

When TABuss suggests a route, the user can respond by validating the result. At the current moment positive user feedback triggers a query run, while negative feedback has no effect.

The level of intelligence is fairly low, but is still higher than in functionalities with direct look-ups, e.g "if val.equals(another)".

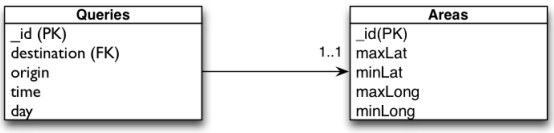

Figure 15: Database tables

3.6.5 BusTUC and Natural Language

To enhance the use of natural language in TABuss, new functionalities which use BusTUC are implemented. Raaum's application only requires a destination as input, which limits the amount of natural language provided to the system. New functionality is firstly the option to switch between the BusTUC syntaxes, defined in section 2.1. While the new syntax assumes that the user wants to depart from one of the closest located bus stops, the standard syntax allows for user defined departure stops. Switching between the two BusTUC syntaxes can be done in the home screen menu.

The second part involves AtB's text messaging service[56]. An SMS query starts with "rute"(route), followed by text, to 2027. This has been incorporated in two ways. If the new BusTUC syntax is chosen from the home screen menu, TABuss uses the closest bus stop to the user's location as the departure stop. If the standard syntax is chosen, the user has to provide both departure and destination input.

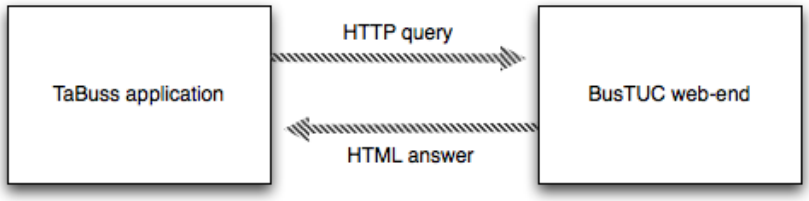

Figure 16: Web end communication

3.6.6 Real-time Functionality

Real-time data can be accessed from the map by pressing a bus stop icon, or through the home screen menu. Both functionalities use the user's location to retrieve and display the n closest bus stops.

The retrieval of a bus stop's ID is done by comparing the chosen bus stop's location with the locations of each of the n closest bus stops. If

[56]https://www.atb.no/spoer-bussorakelet/category228.html

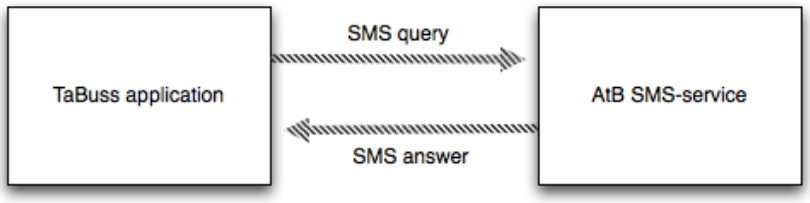

Figure 17: SMS communication

matched, the found bus stop ID is used to extract the real-time ID. The real-time ID is then sent via SOAP to the real-time server, which returns the five next bus departures. The user can also search for bus stops that are not among the n closest, by providing a bus stop name as input. This option also lets the user select which direction to retrieve real-time data for(from/towards the city centre), before a real-time query is sent.

3.6.7 Storage

The external and internal storages are used with SD-card external storage, and a SQLite[57] internal storage database. Favourite strings are stored on the SD-card in text files in a ”favourite” folder. On application start-up, these are retrieved and displayed in the home screen as query shortcuts. The SD-card also stores a text file with all bus stop names. This list is used for the auto-completion functionality in the input text fields.

The SQLite-database contains logged queries and a history of performed real-time searches for bus stop names. The latter allows quick retrieval of previously performed searches.

3.6.8 Display of Answers

The display of query answers is built from scratch. Route suggestions are displayed in a list view, where touch events on a list element trigger a map activity. This map activity shows: The user's location, the location of the selected departure bus stop and a walking route between the two locations. As mentioned in section 2.7, Android does not provide direct access to the Google Maps API, and no native functionality for plotting is available. Our solution is to use KML-files[58] for retrieval of walking coordinates between locations. Plotting is done using an ”ItemizedOverlay”.

[57] http://www.sqlite.org/
[58] http://code.google.com/intl/no-NO/apis/kml/documentation/

3.6.9 Optimisation

To not the map as a main feature is an optimisation, as data traffic is minimised. Other implemented optimisations are according to the limitations with handheld devices. Unnecessary object creations are avoided by using static calls. Existing objects are also used across *activities* if possible. The application will then check whether or not the needed objects exist, and only create new instances if not. As *activities* over time can "pop out" of the *activity stack*[59], initialised objects do not always exist for the entire application lifetime.

Logical computations are put in *asynchronous threads*[60]. This parallelisation makes the start-up of the application faster, and also allows for progress bars to be integrated. Threading is also introduced in the retrieval of real-time data for route suggestions, where a computation time half of the original is achieved. The application creates a new thread for each bus stop to retrieve real-time data for, and sends all queries in parallel. The threads are stopped and removed by a recursive call, when all queries have returned answers.

3.6.10 Shifting Functionality to MultiBRIS' Server

To shift core functionality to a separate server has advantages. During the mentioned Google I/O talks referred to in section 2.9, Reto Meier emphasised the importance of energy conservation in applications because of battery constraints. As back-end computations are shifted to a server, CPU cycles and battery power are saved. Another advantage is the reduced amount of data traffic. In a query involving BusTUC and real-time updates of the departure times, only one query has to be sent to the server. When the application is used stand-alone, a query has to be sent to each.

However, there are disadvantages with the usage of servers. Adding another layer introduces an error source in the communication between the application and the server. BusTUC and the real-time system can be available, but if the server crashes, the application will not receive a response. A server also has to be maintained and updated.

Advantages and disadvantages are further described by MultiBRIS[5].

TABuss benefits from having a modular code architecture, where simple type boolean values control whether to use MultiBRIS' server. Results are either way parsed into dedicated class objects ready for display.

[59] http://developer.android.com/reference/android/app/Activity.html
[60] http://developer.android.com/reference/android/os/AsyncTask.html

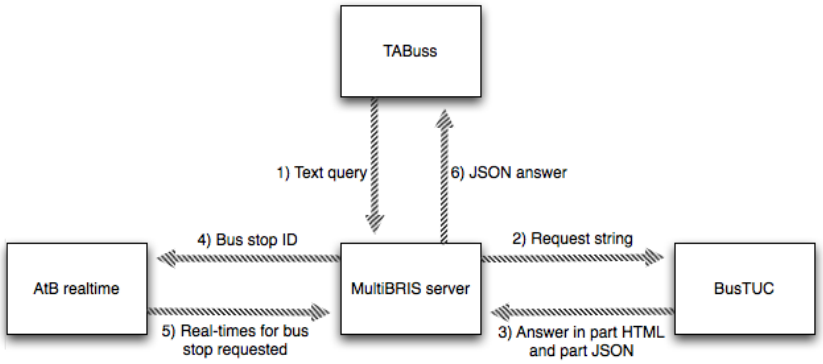

Figure 18: Queries with MultiBRIS' server

4 Results

The following sections describe the development results of TABuss.

4.1 Performance

Raaum's application suffered from poor performance. Queries took up to 40 seconds to complete, which is more than the average user is willing to wait. With the use of MultiBRIS' [5] server, queries rarely use more than 10-15 seconds to return.

4.2 Optimisation with MultiBRIS' Server

An important aspect when developing an application that uses 3G is to monitor data traffic. Table 9 and 10 display average results based on three different tests performed with the application 3G Watchdog[61]. The tested queries used *Gløshaugen* as the departure stop. The first destination was set to *Ila*, the second to *sentrum* and the third monitored data traffic when only real-time data was downloaded.

The results especially differ in the sending of data for the queries to *Ila* and *Sentrum*. As MultiBRIS' server handles all computations, only one request is sent from the application to the server for each query.

Received data is also less, because only one result is returned. For the same queries in the stand-alone application, data was received from both BusTUC and the real-time system. There is also a significant reduction of downloaded data for the real-time query, because only a bus stop ID is sent from the application, and a JSON object is returned.

[61] https://market.android.com/details?id=net.rgruet.android.g3watchdog

It is difficult to replicate the exact same query scenarios when doing comparisons, but as the server handles both computational operations(queries to BusTUC, and real-time updates of departure times), and minimises overhead provided by SOAP messages, we can conclude that less data is sent and received.

The stand-alone application in addition downloads a list mapping bus stop IDs to real-time IDs, on each start-up. This results in an average 400 kb of data, and is in MultiBRIS handled by the server.

Table 9: Data Usage without MultiBRIS' Server

Query	Ila	Sentrum	Real-time
Data sent	5,5 kB	8 kB	4 kB
Data received	6,1 kB	4,4 kB	4,5 kB

Table 10: Data Usage with MultiBRIS' Server

Query	Ila	Sentrum	Real-time
Data sent	2 kB	2 kB	800 B
Data received	3,5 kB	1,5 kB	3,5 kB

4.3 Screenshots

4.3.1 Screenshot Descriptions

1. Start menu where text buttons represent shortcuts stored on the device's SD-card.

2. Application menu. Menu elements are translated to English below.

3. The answer screen with results from a HTTP query with the new syntax. The displayed routes are the results of a BusTUC query with real-time updated departure times. In parenthesis, walking distance

Table 11: Translation of menu elements

Norwegian	**English**
Legg til ny	Add a new bus stop shortcut to the home screen
Logg	Logged queries
Gå til kart	Proceed to map
Innstillinger	Settings
Om denne appen	About this application

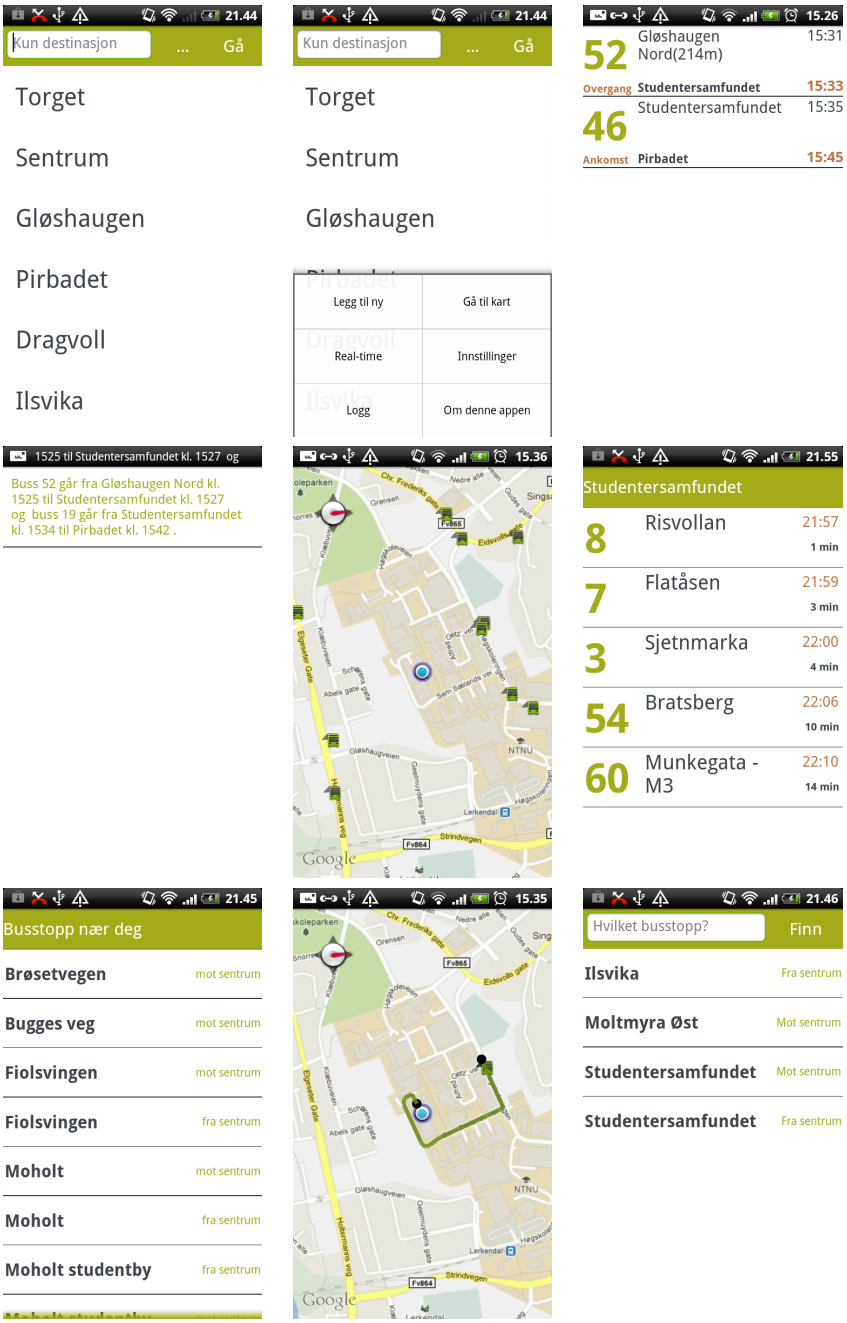

Figure 19: From top left: (1)start screen, (2)menu, (3)answer screen, (4)text answer, (5)map for real-time. Displaying user location and closest bus stops, (6)real-time for stop,(7)list of real-time stops, (8)Walking route, (9)Bus stop search

to the bus stop is shown. ”Overgang” indicates the resulting route suggestions include a transfer.

4. The answer screen with results from a text message query with the standard syntax. Both the text message and HTTP functionality with standard syntax, will output results to this answer screen.

5. Map displaying user location. The closest bus stops are represented by clickable bus stop icons.

6. Result of a real-time data query. The query is either initiated by menu access, or by a bus stop icon press.

7. List of the closest bus stops to the user's location, accessed from the menu. On registered clicks, real-time data is downloaded for the selected bus stop. Each element also displays route direction, either towards or away from the city centre.

8. Map displaying walking route to a departure bus stop suggested by query results.

9. Search functionality for bus stops not in range of the user's location. If the search returns a bus stop, real-time data can be viewed. The list of elements below the input field contains recently searched bus stops. These are stored in a SQLite database.

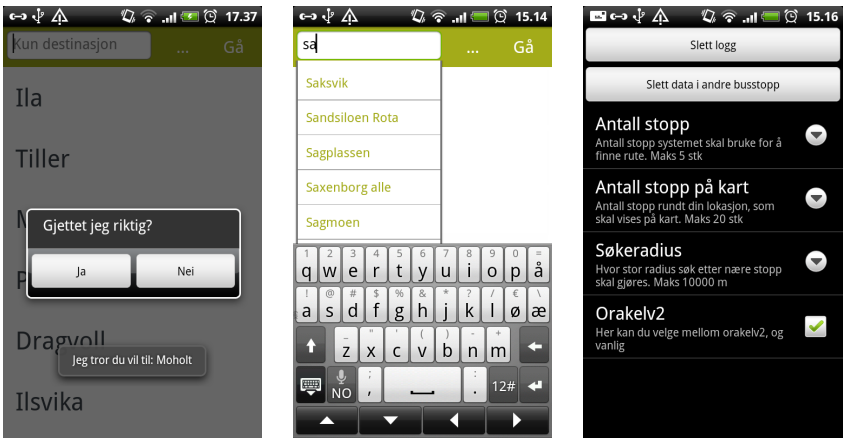

Figure 20: From left to right: (1)guess based on context, (2)autocomplete, (3)settings

1. Suggestion from TABuss, based on stored cases. The bottom pop-up suggests a route, while the above dialog prompts for validation.

2. Autocomplete suggestions, retrieved based on the two letters entered in the input field. If clicking a suggestion, a query is run with the chosen suggestion as the departure stop.

3. The settings screen, where options are: delete logs, adjust number of bus stops to be included in queries with the new syntax, adjust number of bus stops to be displayed on the map, adjust search radius for bus stops and an option to switch between new and standard BusTUC syntaxes.

4.4 System Testing

TABuss has been tested before, during and after development. Most errors affecting core functionality were fixed during the testing of Raaum's application. Other errors were detected during runs, by running queries with different inputs, and also by adjusting the settings parameters. Testing was mainly done with the "Eclipse Debugger[62]", which displays error traces if exceptions occur. Our goal regarding error rate was not to achieve zero percent. With the project's time limitations, this would be too optimistic. Instead we decided to do testing continuously during implementation, fix what we could, and identify possible error sources for future reviewing. An example of a performed test was during the implementation of the error messages a user can be presented. Different scenarios were designed for the application to throw exceptions: Unmount of the SD-card after application start-up, disconnection from a network and lost location fix.

Testing without the Eclipse debugger was done by travelling routes suggested by the application. Detected exceptions were stored on the test devices' SD-cards, and reviewed when connected to a development machine.

4.5 User Testing

A simplified user test was performed to get feedback of TABuss' functionalities. An extensive user test was not conducted because of time limitations, and issues with permission for public release of the application.

All of the test subjects were Trondheim inhabitants, and experienced bus travellers.

4.5.1 General Opinion

The general user opinion indicated that the application was easy to use. However, some users experienced difficulties during installation, as their devices did not meet the original SDK requirements(2.3).

[62]http://www.eclipse.org/

It was clear that the users appreciated our prioritising of user interface. Positive feedback was received on both the colour combinations and the layout. Most users preferred the application functionalities detached from the map, and feedback suggested the map should only be an add-on.

Users found the query functionality to be useful. The main functionality with the new BusTUC syntax was seen as interesting. Accustomed to BusTUC with standard syntax, not having to provide the same amount of text was a time-saver. It was subsequently easier for the users to blame the system if an erroneous answer was returned, as user input was limited.

The real-time data functionality for the closest bus stops was a functionality found quick to access and use. This especially applied to the real-time functionality accessible from the home screen, as this required less navigation than through the map.

4.5.2 Suggestions

All users requested a more extensive feedback from the system, when errors occurred. Errors such as: A missing internet connection and no location fix, had up to this point been covered by a general error feedback.

For the query functionality, users requested the possibility to use the standard BusTUC syntax for queries not involving the closest located bus stops. This was not an option at the current development stage. Another suggestion was a ”settings”-screen, allowing the user to set properties such as the number of bus stops to use in queries.

4.5.3 Implemented Suggestions

When solving the installation problems some users had, a memory bug was discovered on Android 2.2. The loading and parsing of bus stop lists needed to be re-implemented, as a memory overflow occurred. The list used in Raaum's application[17] contained 498 elements, while the new lists each contains over 1000. After researching this error, a possible error source was found on a debugging forum [63]. One forum user posted that he did not get his application to work with >505 elements. If correct, this explains why this was not detected during Raaum's project. A bug report had been filed to Google regarding this issue. The re-implementation consisted of manually parsing the XML-files, instead of using Android's built-in parser.

For the system feedback request, additional error messages were added. This included checks for internet connection, mounted SD-card and location fix. The usage of the standard BusTUC syntax was also implemented. The last added suggestion was a settings functionality, which lets the user set different properties within given boundaries.

[63] http://stackoverflow.com/occuring

4.5.4 Conclusion

The user's opinions were divided in the choice between the BusTUC query functionality and only real-time functionality. A possible reason may be speed, as the BusTUC query functionality during user testing was not optimised. Another reason may be the limitations some users experienced with the new BusTUC syntax.

It is difficult to draw a concise conclusion after a narrow user test, but the feedback we received from the target users was valuable. We received implementation suggestions, information on errors and an indication that TABuss suited the needs of bus travellers.

5 Discussion

5.1 Advantages

We are satisfied with the development process, including the learning of new technology aspects involving Android. There were no major problems during development, and research progressed in parallel. The implementation of new features was done in iterations, resulting in a functional application for every demo.

5.2 Improvements

Development was delayed due to unavailability of the source code from the Raaum's application[17]. In retrospect, we should have been more aggressive towards retrieval. Due to tragical circumstances there was also a switch of supervisors. This happened at an unfortunate time, as decisions had not yet been made regarding task descriptions. Whether this lead to less functionality being implemented is uncertain, as development progressed rapidly when the source code eventually was received.

The shifting of functionality to MultiBRIS' server [5] caused some duplicate programming, as MultiBRIS began the implementation of server functionalities in parallel to our development. While not visible in the running of TABuss, parts of the source code providing the same functionalities, is written separately by both groups.

5.3 Answer to Research Questions

Reviewing the defined goals, we claim that all have been fulfilled.

Raaum's application[17] has been thoroughly reviewed and tested, and all obvious bugs have been fixed. This was done before any new functionalities or major changes were implemented. While the time estimates regarding duration was exceeded, the goal is still considered to be reached.

None of the research papers regarding natural language directly affected development, as the purpose was to facilitate for future work. The articles found and analysed represent a good foundation.

The implementation of a new user interface was an important goal to reach. The resulting user interface is satisfying, and the feedback received from users support the graphical choices made. Doing design with usability in mind was given a considerate amount of resources, as it was important that an acceptable suggestion was finalised within given time.

For the actual development, all planned functionalities involving usage of real-time data were implemented, including searching and storing possibilities.

The standard syntax for BusTUC queries was also implemented. Hopefully, users will prefer the new syntax, but both syntaxes are still available.Together with the text messaging service, covering usage without an internet connection, natural language queries can now be sent to BusTUC in three different ways.

The server shifting proved to be successful. While TABuss still works as a stand-alone application, computational gains are achieved, especially during start-up. The loading of bus stop lists has been reduced, and the loading of the Real-time ID list has been removed. The time estimated for shifting functionality was about the same as the actual time spent. The shifting is considered to be permanent at this point. Future development should continue to utilise the server, as more functionality developed client-side would put further restraints on the underlying resources.

It was concluded during research of existing solutions in section 3.2, that *Bartebuss* and *Alf's ByBuss* were comparable applications to TABuss. As Rune M. Andersen has been available as a resource through the development process, *Bartebuss* has mainly been used for comparisons. Compared to *Bartebuss*, TABuss' functionalities are more focused on user location and context awareness. In our opinion, intelligence is what separates TABuss from *Bartebuss*, and also from the other test subjects for Trondheim. We claim that in order for an intelligent bus route information application to actually "be intelligent", the natural input source is context data. Whether TABuss can be classified as "better" is unsure, as *Bartebuss* has been developed over a longer period of time, and been through more extensive user testing. Still, we feel TABuss represents a more complex approach, with market potential, as no other applications have the exact same functionalities.

We discussed in Section 2.9 the advantages and disadvantages with native and web development, and stated that both of the project members prefer native. In retrospect we are satisfied with our technology choice. Compared to *Bartebuss*, a technology difference is in the storage functionalities. For Bartebuss to work cross-platform and also through a regular

browser, ”web storage” through ”local storage” is used[64]. The size limit of local storage depends on which browser is used, but is no bigger than 10 MB(Internet Explorer). TABuss uses the devices' external storage, where the size limit depends on the size of the mounted SD-card, which normally can store gigabytes of information. While not necessary in the current version of TABuss, future extensions could need more storage space than 10 MB. The storage limitation of local storage also affects the iOS version of Bartebuss, where the internal storage optimally is used instead(external storage not available).

The map problems in web applications deployed on the Android platform are avoided in TABuss, where the map is much more responsive. Native development also allows for pinch zooming, which is an important feature when navigating.

The use of *activities* utilise the devices' screen sizes, because the same information does not have to be displayed at all times(can instead navigate between activities). TABuss also binds the devices' buttons to functionalities such as the home screen menu, which also contributes to minimise the amount of displayed information.

Although web applications can be deployed on multiple platforms, native applications provide, in our opinion, the best user experience for Android and our domain. We prefer to rather develop a competitive application for a specific platform, than to deploy a ”working” solution to more. It has to mentioned that this is given today's web application performance on Android. Future SDK updates will benefit web application development and improve the browser rendering[65]. The problem is that older devices will not receive these updates, and it will also take time for newer devices to get them. The releases to newer devices almost always have to wait until the different manufacturers have adapted their own distributions. Developers will then have a dilemma on which SDK versions to target, and which users to exclude.

TABuss does not represent a complete solution or the ”holy grail” for intelligent route information applications. It represents a contribution, and a motivation for others to do future development. We have illustrated possibilities with mobile development and artificial intelligence and, together with MultiBRIS' server[5], developed a working system. TABuss differs from other applications providing bus route information in Trondheim, and has not at this point any competitors regarding the level of artificial intelligence.

[64]http://en.wikipedia.org/wiki/Web_Storage

[65]http://www.sencha.com/blog/galaxy-nexus-the-html5-developer-scorecard/

5.3.1 The Future of The Application

The future of the application is to be decided by the department. The concept of an intelligent bus route information system will most likely continue to be pursued during next semester's masters thesis. However, the direct involvement of TABuss is unsure.

6 Future work

6.1 TABuss

TABuss has not been thoroughly user tested after its recent updates. A detailed user test will elucidate problems concerning both aesthetics and functionality, better than random bug reports from just a few users. Optimally, this should be done before TABuss is uploaded to the Android Market.

A functionality that is missing is dynamic switching from using MultiBRIS' server to using BussTUC and the real-time server directly. The intention is to switch if MultiBRIS' server experiences problems. Basic implementation should be relatively easy, as all the functionality exists for manual switching. More advanced implementations could be similar to the *Spectra* system developed by Flinn, Soyoung Park and Satyanarayanan[7]. *Spectra* monitors resource usage during operations, and uses this to decide whether to perform operations locally, with a remote server or through a hybrid solution. In TABuss this would mainly concern the queries' time consumptions, and a choice to switch to stand-alone computations if the server is too busy.

The bus stop lists should reside on the server. Data should be downloaded to the application on start-up, and only if there are any changes since the last startup. As bus stop IDs and bus stop names are regularly updated by AtB , managing the lists on the server prevents the user from having to download a new version of TABuss each time a change occurs in the list.

We were made aware of an SMS API text messaging service located at IME [66]. Shifting to this could be an option for students to save some money on behalf of the department during debugging.

The functionality for guessing a user's intended destination is incomplete, and only a little testing has been performed. Continued development should strive for getting correct suggestions most of the time. A high success rate is both intelligent and user friendly. In order to achieve a high success rate the "simplified" case-based reasoning described in Section 3.6.4 will have to be extended.

[66] http://boost.com

To further utilise the advantages of native development, widget functionality could be implemented. An idea is to have a widget displaying information on the closest bus stop to the user's location. If the user chooses to access the widget, the widget itself could provide some information or trigger the start-up of the actual application.

Another interesting field is Near Field Communication(NFC)[67]. NFC is based on RFID standards, and can be used to set up ad-hoc[68] networks. Typical usages include instant messaging services and games. A usage involving TABuss could be to integrate an RFID tag in every bus stop, and trigger display of real-time data when the user approaches.

6.1.1 Known Bugs

When downloading of real-time data the returned JSON objects from AtB's real-time system sometimes contains nodes with the wrong date, while the time is still correct. This does not affect the display of departure time, only the display of additional information on minutes to departure. The value grow erroneously large, as the wrong dates lead to wrong calculations. The real-time system is also unstable around midnight as it does not return JSON objects formatted similarly to the ones returned during daytime. In detail, the nodes containing departure times are missing. Both of these bugs are present in the real-time server, and will have to be addressed by AtB.

Another bug is in the retrieval of the closest bus stops, and the distance to them (in metres). The built-in Android algorithm for calculating distance between two locations returns the air distance in metres. An optimal solution would be to use the walking distance: air distance disregards physical objects that may be blocking the way. A possibility is to use KML-files, as described in Section 3.5.2.

The application has (on a few occasions) continued to run in the background when it should have exited. This may indicate that there is a need for a more careful exit process.

For the query functionality with BusTUC, TABuss sometimes cannot return an answer when the user is located in a specific area in the city centre. This problem occurs on the server, and is handled as any other exception by TABuss.

6.2 Future Research

This section identifies future research areas of FUIROUS and TABuss.

[67] http://en.wikipedia.org/wiki/Near_field_communication
[68] http://en.wikipedia.org/wiki/Wireless_ad-hoc_network

6.2.1 Speech Processing

IDI has a speech-extension to BusTUC(*TeleBuster*),but there is no certainty to whether future development of this extension provides the best possible solution.

TravelMan(2007)[24] (2006)[23] is an interesting system because of it's speech processing and route guidance. A goal could be to create a similar prototype, and compare with other implementations including *TeleBuster*.

Architecturally, it is important to decide which parts of the system should perform the speech processing. Integrating the functionalities on Multi-BRIS' server will minimise application computations and contribute to a modular solution. Application side implementations also has its advantages, such as sending small parsed texts across HTTP instead of sending large sound files.

While TABuss has explored the opportunities of involving smartphones, another area of research is the the extension of functionality to users with only regular cell phones. TABuss already utilises SMS for communication with BusTUC, and a speech recognising module could involve a simple calling interface. The predecessor to *TravelMan*, *StopMan*[25](2006), used a calling interface to provide route information, and could together with IDI's *TeleBuster* be a natural starting point.

6.2.2 Context Awareness

TABuss uses location data as context input. An extension is to use more sensors than only the location sensor. Raento, Oulasvirta, Petit and Toivonen(2005) developed the system *ContextPhone*[18]. *ContextPhone* uses four sensors: location, user interaction, communication behaviour and physical environment. This means that besides from location information, *ContextPhone* monitors: what actions the user performs, calls and SMSs and surrounding devices.

For TABuss, this sensor information could be used to introduce context awareness to the user interface. The age differences between potential target users is large, and an adaptive user interface could be a solution. The user interface could through sensors track the user's actions, register some trends and then adjust visibility and availability accordingly. An alternative may be in the direction of the work performed in 2010, by Kolekar, Sanjeevi and Bormane[12]. They proposed an adaptive user interface solution with the use of feedforward artificial neural networks[69] and backpropagation[70] to learn the user's behaviour.

The tracking of user trends could also be used to perfect route suggestions. People of different ages have different levels of mobility, and have

[69]http://en.wikipedia.org/wiki/Artificial_neural_network
[70]http://en.wikipedia.org/wiki/Backpropagation

different walking speeds. This has been addressed by Vieira, Caldas and Salgado(2011), in their proposed system *UbiBus*[26]. UbiBus considers different people's and vehicle's mobility, and other factors than can affect a bus departure. An interesting idea is for AtB to contribute to such functionalities in order to improve route suggestions. Buses have installed cameras, and could be used to monitor how crowded a bus is. This could prove beneficial for handicapped people, or people with small children, who need seats or at least clear floor area.

6.2.3 Intelligent Computations

In Section 6.1, the *Spectra* system was mentioned. For TABuss to use a similar approach, resource usage has to be monitored. An algorithm implemented on MultiBRIS' server could then be used to make the actual choice, on whether TABuss should run queries as a stand-alone application, or to use MultiBRIS' server.

It is also possible to implement computations based on monitoring results client-side, in TABuss. Monitored data for runs could be stored on the device as cases, and a case-based reasoning implementation could retrieve the best matching ones. An artificial neural network approach is also possible, where the system makes decisions based on learnt information. Training could be done over a number of runs, for TABuss to learn which operations to perform locally, and which to perform through MultiBRIS' server. An example is a standard BusTUC query: if the monitored battery power is low, TABuss should opt for server computations to minimise CPU cycles. However, if the server is busy(caused by a high traffic load) TABuss has to view earlier runs in order to find the best solution.

6.2.4 Future Extensions of TABuss

A future extension could be to integrate TABuss into a tourist application. The *Trondheim Guide*[71] is an intelligent travel guide which already implements some bus information. This information is limited, and we could not find any information on arrival/departure times. Another alternative is *City Explorer*[72], which is a framework for city exploration.

6.3 BusTUC

Future work on BusTUC involves researching other solutions for intelligent route information. As BusTUC is the only available candidate in Trondheim, there are no available comparisons. One specific task would be to do

[71] www.trondheim.no/app

[72] http://www.sintef.no/Projectweb/UbiCompForAll/Results/Software/City-Explorer/

research on similar systems, and to develop comparable prototypes. Expansion of BusTUC outside of Trondheim, to cities of different sizes and number of inhabitants, is also an exciting option. If BusTUC turns out to be the best solution, a goal could be to establish it as a standard for bus route information in Norway.

6.4 Real-time

AtB has planned the implementation of SIRI[73]. SIRI is a CEN standard[74] XML-protocol for the retrieval of real-time data. In Oslo, Trafikanten has made the *StopMonitoringRequests*[75] publicly available through a JSON API.*StopMonitoringRequests* offer the same functionalities as AtB's real-time system offers today, with real-time departure times of buses. By using a JSON API the overhead provided by SOAP-messages is avoided, which means less data needs to be transferred.

In the "Experts in teamwork"[76] subject in 2011, one of the project members participated in a project aimed towards the use of the SIRI standard. We were then made aware of AtB's plans, but also that delays already had occurred, and most likely would continue to occur. Keeping these delays in mind, TABuss' real-time functionalities should be more modularised. As AtB had no accurate answer for when the SIRI implementation would be finalised, a modularised application could benefit from a "plug-and-play" solution. TABuss would then not have to re-implement functionalities when the SIRI implementation is finalised, but simply swap.

An ultimate goal should be for all bus companies to use the same standards. This would aid the development of bus route information systems, because adaptation to more cities would be simplified. When AtB has implemented the SIRI standard, real-time data for two major cities in Norway use the same standard. In addition has Bergen begun a real-time data trial period, with GPS trackers installed on trams[77]. It may therefore be reason to believe that future real-time data implementations in other cities will follow the SIRI standard, which would simplify a future version of TABuss to be adapted to several cities.

[73] http://www.kizoom.com/standards/siri/
[74] http://www.cen.eu/cen/pages/default.aspx
[75] http://www.kizoom.com/standards/siri/schema/1.4/examples/exs_stopMonitoring_request.xml
[76] http://www.idi.ntnu.no/grupper/sos/eit2010/
[77] http://labs.trafikanten.no/2011/3/1/sanntid-paa-bybanen-i-bergen.aspx

7 Acknowledgements

We would like to thank our supervisors Björn Gämback and Rune Sætre for guidance and support. Rune has especially been a valuable resource to have, both because of his knowledge of the BusTUC system, and his general interest in mobile application development. We would also like to thank our beta-testers: Jirka Konietzny, Trond Bøe Engell, Jostein Klakegg, Ola Hast, Marita Gjerde, Håvard Axelsen and Morten Fornes.

References

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AICom - Artificial Intelligence Communications*, 7, 1994.

[2] Tore Amble. Bustuc: a natural language bus route oracle. In *Proceedings of the sixth conference on Applied natural language processing*, ANLC '00, pages 1–6, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[3] David L. Applegate. *The Traveling Salesman Problem: a Computational Study*, page 1. Princeton University Press, 2006.

[4] 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.

[5] Trond Bøe Engell and Runar Andersstuen. Multibris: -a multiple-platform approach to the ultimate bus route information system for mobile devices. Master's thesis, NTNU, December 2011.

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

[7] 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.

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

[9] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to "a formal basis for the heuristic determination of minimum cost paths". *SIGART Bull.*, pages 28–29, December 1972.

[10] Mike Hazas, James Scott, and John Krumm. Location-aware computing comes of age. *Computer*, 37:95–97, February 2004.

[11] R.E. Kalman and R.S. Bucy. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[12] S.V. Kolekar, S.G. Sanjeevi, and D.S. Bormane. Learning style recognition using artificial neural network for adaptive user interface in e-learning. In *Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference on*, pages 1 –5, dec. 2010.

[13] S.D. Maclean and D.J. Dailey. Mybus: An advanced public transportation system based on the us tcip standard. *Proceedings of the Seventh Annual World Congress on Intelligent Transport Systems*, November 2000.

[14] S.D. Maclean and D.J. Dailey. Real-time bus information on mobile devices. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 988 –993, 2001.

[15] J Pascoe. Adding generic contextual capabilities to wearable computers. *International Symposium on Wearable Computers*, 2:92–99, 1998.

[16] John A. Quarantillo. Gui kisses: Tips and strategies for interface design. *SUGI*, 24, April 1999.

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

[18] Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen. Contextphone: A prototyping platform for context-aware mobile applications. *IEEE Pervasive Computing*, 4:51–59, 2005.

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

[20] N. Ryan, J. Pascoe, and D. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications and Quantitative Methods in Archaeology (CAA 97), Oxford*, 1997.

[21] Bill N. Schilit and Marvin M. Theimer. Disseminating active map information to mobile hosts, 1994.

[22] Robert J. Szczerba, Peggy Galkowski, Ira S. Glickstein, and Noah Ternullo. Robust algorithm for real-time route planning. *IEEE Transactions on Aerospace and Electronic Systems*, 36(3), July 2000.

[23] Markku Turunen, Jaakko Hakulinen, and Anssi Kainulainen. Evaluation of a spoken dialogue system with usability tests and longterm pilot studies: Similarities and differences. In *In Proceedings of Interspeech 2006*, 2006.

[24] Markku Turunen, Jaakko Hakulinen, Anssi Kainulainen, Aleksi Melto, and Topi Hurtig. Design of a rich multimodal interface for mobile spoken route guidance, 2007.

[25] Markku Turunen, Topi Hurtig, Jaakko Hakulinen, Ari Virtanen, and Sami Koskinen. Mobile speech-based and multimodal public transport information services, 2006.

[26] Vaninha Vieira, Luiz Rodrigo Caldas, and Ana Carolina Salgado. Towards an ubiquitous and context sensitive public transportation system. *International Conference on Ubi-Media Computing*, 0:174–179, 2011.

[27] Alf Inge Wang, Carl-Fredrik Sørensen, Steinar Brede, Hege Servold, and Sigurd Gimre. Development of location-aware applications the nidaros framework, 2005.