A Multiple Platform Approach to Building a Bus Route Information System for Mobile Devices

Runar Andersstuen **Trond Bøe Engell**
{runaa,trondboe}@stud.ntnu.no

Rune Sætre **Björn Gambäck**
{satre,gamback}@idi.ntnu.no

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

Abstract: The paper describes a multiple platform-based approach to creating a bus route information system for mobile devices. The system is context aware: users only need to tell the system (in natural language) where they wish to go, and the system takes care of the rest. The users are presented with a list of possible routes they can take to reach their desired destination. The results are also shown on a map that makes finding the bus stops very easy.

In order to make the system available to as many users as possible, the architecture is client-server-based and relies on technology standards that are widely accepted and implemented, making it easily adaptable to new platforms. The application can be run on multiple platforms, with a minimal amount of calculations needed on the client side. The amount of data transfer between server and client is also kept to a minimum.

The ability to run on multiple different platforms is achieved using technology such as HTML5, PhoneGap and Sencha Touch. The client's functionality includes a search function and a map view, as well as the ability to use bookmarks. The server handles most of the business logic and communicates with external services such as the natural language processing back-end and the server for real-time bus departure information updates.

1 Introduction

The worldwide smartphone market has expanded immensely during the last few years. There were 440 million mobile devices sold by vendors in the 3rd quarter of 2011 [Pet11]. Of these, 115 million were smartphones. This equals a market share of 26.1%. The market share has increased continuously during the last few years. Companies like Apple, HTC and Samsung that have focused on developing smartphones, have gained large parts of the market share. Other companies, like Nokia, that are big on mobile phones, seem to be on a negative trend. As smartphones take over the market, the need for mobile-enabled content and services increases. Unfortunately, the huge amount of available devices has split the market into several mobile technology platforms. Leading platforms like Android, iOS (Apple), and RIM are all based on different operating systems and code languages.[1] Software developers need to make choices on which platform to support and then learn the native language of that platform. If they want to focus on several platforms and reach out to a larger audience, duplicate efforts are needed to implement specific software on each platform and keep maintaining each code base separately. Consequently, application development time can be immense.

One of the reasons why smartphones are popular is that people always bring their phones with them, wherever they go. The mobility of smartphones opens up for new use-cases where stationary PCs cannot compete. Uncertainty in the highly competitive commercial mobile market space causes questions to appear when approaching the challenges of cross-platform publishing: Which platforms will succeed? What resources and knowledge are needed to make a sensible decision? Fortunately, there is a way around this issue: multi-platform development. The big advantage here is a single code base, which reduces application development time greatly. There are several ways of developing multi-platform software. Determining which strategy that is most suitable can be a challenge, and it is certain that the project requirements must be the primary decision-factor. The main alternative strategies to create multi-platform applications are the following:

Web-based applications make use of HTML5, Javascript and Cascading Style Sheets (CSS) to create mobile websites that aim to look and feel like a native mobile application. Web applications can use JavaScript frameworks, such as Sencha Touch and jQuery Mobile, that are solely designed for mobile development, to replicate mobile user interfaces.[2] Web applications can be conveniently run in a web browser and are therefore already multi-platform, since most mobile device today are equipped with web browsers. One disadvantage for web-based applications is that they have only limited access to device-specific features. Also, they cannot be uploaded to application stores, like the Android Market or the iOS App Store, which will have a negative effect on the availability of the product.

Proprietary Middleware. Applications can also be based on web services such as Red Foundry.[3] Developers get access to a web interface where an application is graphically created by selecting a set of prebuilt modules. When all the modules that provide the necessary functionality have been picked, the service builds a native application which can be submitted to an application store or market. The advantage of this strategy is that the developer does not need any specialist knowledge or programming experience to create applications that look good and perform well. The drawbacks are that the proprietary services often are expensive and that the design and functionalities are limited to what is offered by the service.

Native Applications are written in a specific code language and designed to run in a specific operating system. The main advantage of native applications is that they work as intended by the operating system developers. Device features like sensors, contact lists and storage are easily accessed directly, and the libraries offered by the application programming interface (API) are optimised for the specific operating system.

[1]See `http://www.apple.com/ios`, `http://www.android.com` and `http://www.rim.com`.
[2]Available at `http://www.sencha.com/products/touch` resp. `http://jquerymobile.com`.
[3]`http://www.redfoundry.com/`

Hybrid applications are written as web applications, using coding technologies such as HTML5, CSS and JavaScript. The web applications are then wrapped by one of the available "multiple phone web-based application frameworks" in order to emulate native behaviour. Device features like sensors, contact list and storage are provided by these platforms. Unlike the other alternatives that are confined to browsers and have limited functionality, hybrid solutions form a strong strategy for multi-platform development [Pad11, Chr11]. Developers get a greater control over application design. They use one single code base, but still get access to device features.

In a purely server-based system, all the business logic[4] resides on the server. Having the business logic on a server provides several benefits. First, it saves the client from heavy computations, which is desirable because saved CPU cycles means saved battery power [FZ94]. Second, if all the business logic is handled on the server side, less data needs to be transferred to the client. Finally, since there is one place where all the business logic is handled, optimisation through resource sharing and information caching is easier to implement. Updates are also made easier, for instance, if one of the external service providers decides to alter how their service is accessed, only one central update is needed. If the business logic existed on the client, all existing applications would need to have their code updated to handle the new service change.

In order to reach many users on different platforms, it is essential that the system relies on technology standards that are widely accepted and implemented. The paper introduces a prototype system is called MultiBRIS: a Multi-platform Bus Route Information System. MultiBRIS gives access to a natural language bus route system for Trondheim. The system is called BusTUC [Amb00] and was previously only available via the web or SMS, but through MultiBRIS, it can now be accessed from multiple different smart-phone platforms. The MultiBRIS work is based on a previous application developed specifically for the Android platform [Raa10]. MultiBRIS extends and generalizes this into an application for multiple platforms, supporting real-time bus route information, location tracking and context awareness. All functionality from the previous native Android application has been implemented, and the multi-platform application gives the user the look and feel "illusion" of being a native application. The hybrid application strategy worked out as planned and gave the benefits from using the multiple platform approach we were looking for, making it possible to successfully create one application which can run on both Android and iOS devices without the use of any platform-specific code.

The paper first describes other related bus route applications in Section 2, and then gives an overview of the multi-platform application in Section 3. The benefits and improvements rendered by the client-server solution are discussed in Section 4, while Section 5 concludes and points to areas of future research.

[4]'Business logic' is a term relating to the functional algorithms that handle the information exchange between a database and a user interface. In this paper, the term will be used to refer explicitly to the part of the system that makes the actual computations and calls the external services.

2 State of the Art

To give an overview of the technology and functionality available in current smartphone route guidance applications, this section first reviews two topical systems, Google Transit and OneBusAway, and then makes a thorough comparison of the various smartphone-based bus route information applications presently available in the city of Trondheim.

Google Transit is a general public transportation planning tool integrated with Google Maps. It essentially consists of two parts: the *Google Transit Trip Planner* (GTTP) which relates to the consumer of the service, and the *General Transit Feed Specification* (GTFS) which is used by data providers (typically public transportation agencies) to feed the Google Transit Service with data [Mor09]. GTTP lets users pose queries to the transit system in three different forms: by address, by a location name with directional indicators (NE, NW, SE, and SW), or by GPS coordinates. All queries are accompanied by date and time, and query types can be combined. A result from GTTP displays both text and directional lines on Google Maps. Google Transit offers a good way for users to directly interact with the service. The drawback is the lack of APIs for external developers, making it impossible for them to use the data in their systems. However, GTFS provides a good starting point as to what data is needed in order to make a good public transport system.

OneBusAway is a set of transit traveler information tools developed for providing real-time arrival information to Seattle area bus riders [FWB10]. It includes a trip planner, a schedule and route browser, and a transit-friendly destination finder. The project has concentrated on tools for providing real-time arrival information and includes functionality supporting location sensing. An iPhone application was created first, to exploit its localisation framework and built-in multi-touch map support. Later this was generalised in a JavaScript-based experimental multi-platform web application for real-time arrival information. A OneBusAway user study provided valuable user feedback for such applications, including that the users want a bookmark functionality, enabling them to tap a bookmarked destination at any time and receive route suggestions to it from their current location.

To find other bus route information applications in Trondheim, both the web, the Android Market and the Apple App Store were searched. Table 1 compares the functionality of the applications found on October 5th, 2011. The applications range from simplistic to more sophisticated with lots of functionality, as indicated by the comparison chart.

Google Transit differs from the other systems by being purely server-based. All business logic resides on the server. The client, which is a web browser, only handles the display of information. For the Trondheim applications, it is interesting to note that the web-based BarteBuss that has been created by means of HTML5 and JavaScript is the most feature rich and well-working application of them all. This hybrid strategy, using HTML5, JavaScript, CSS, and a deployment technology can be used to easily create a client prototype that works on several platforms. As discussed above, a server-based solution in addition offers several other benefits, and is thus chosen here.

	Buss-Orakel	Barte-Buss	Buss-tider	Alf's Bybuss	Buss-droid	BusApp Tr.heim	Buss-ruter	Buss-øye
Platform	A/iP	web	A/iP	A	A	A	A	iP
Multi-Language	Yes	No	No	Yes	No	Yes	No	No
Cost	No	No	No	No	No	Yes	No	No
Favourites	No	Yes	No	No	Yes	No	No	Yes
History	Yes	Yes	Yes	Yes	No	Yes	No	Yes
Route download	No	No	No	Yes	No	No	Yes	No
Map function	No	OSM	GM	GM	No	GM	No	GM
Closest bus stops	No	Yes	No	No	No	Yes	No	Yes
Uses GPS	No	Yes	Yes	Yes	No	Yes	No	Yes

Table 1: Comparison of the bus route information applications available in Trondheim
A=Android, iP=iPhone, web=HTML5, OSM=OpenStreetMap, GM=GoogleMaps

3 System Overview

This section describes the prototype server developed during this project. First, the system services are introduced and then a description of the technologies is given. The importance of a multi-platform application has been explained earlier. The languages HTML5, CSS and JavaScript make it possible to create a multi-platform solution. In order to reach out to all users, the application also needs to be available from a variety of application stores which provide opportunities for publishing, advertisement and collecting fees. Hybrid applications are designed to give the advantages describes above using deployment technologies. The disadvantages of the hybrid-solutions will dissipate as the deployment technologies, along with the browsers on the devices, mature and become more robust.

The server offers three distinct services. Figure 1 shows a block diagram of the system. The "main service" effectively replaces all the business logic implemented in the previous

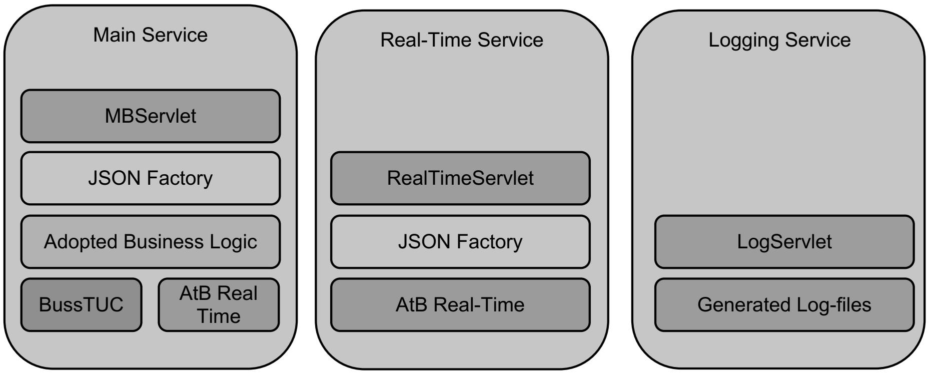


Figure 1: Block diagram of the server system
(MB = MultiBRIS, JSON = JavaScript Object Notation)

Android application [Raa10] and is described in Figure 1. The system also includes a "real-time service" where the client can send a bus-stop ID to the MultiBRIS server and get a list of the exact time for the next five buses arriving at that stop. Finally, the server provides a "logging system service" (for debugging purposes) which is easily accessible through a web browser.

3.1 The Main Service

Figure 2 shows a complete interaction diagram for the main service.

The HTML returned from the BusTUC natural language system contains both a textual answer and a JSON object, but TABuss only uses the JSON object. Calculating the distance between two GPS coordinates is done on the server. The distance calculation is not as trivial as just using the Euclidean distance, because of the approximately oblate spheroid shape of the Earth. The distance (between two GPS locations) is therefore based on Vincenty's inverse formula [Vin75], using the World Geodetic System (WGS 84) standard.

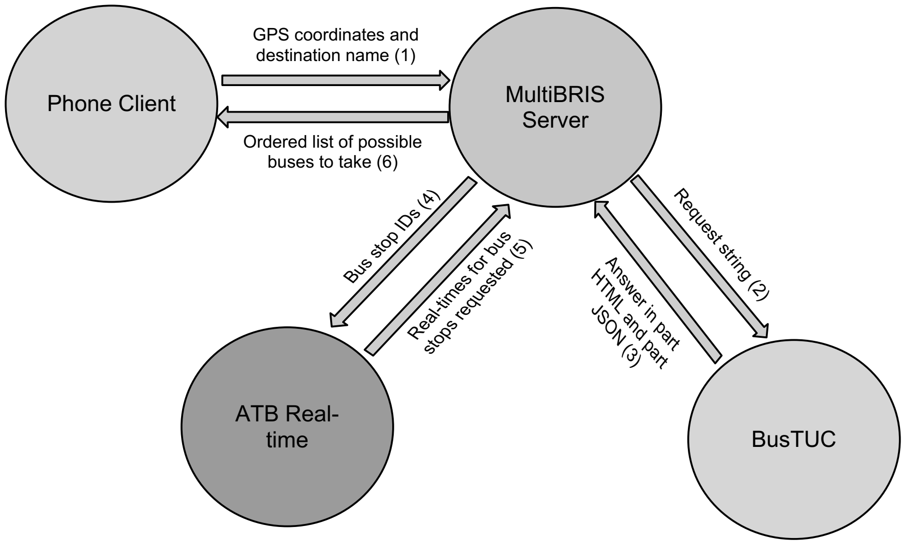

Phone Client
GPS coordinates and destination name (1)
MultiBRIS Server
Ordered list of possible buses to take (6)
Bus stop IDs (4)
Real-times for bus stops requested (5)
Answer in part HTML and part JSON (3)
Request string (2)
ATB Real-time
BusTUC

Figure 2: Main service overview.
1: The client sends a desired destination and its current location to the MultiBRIS server. The server looks up a fixed number of bus stops near the client's location.
2,3: When the bus stops are found, a query is sent to the BusTUC web-interface which responds with a set of the next scheduled bus routes from the clients destination to the target.
4,5: The MultiBRIS server then updates these routes with real-time departure times. The real-time update is done by contacting a real-time system provided by AtB, the current public transportation service provider in Trondheim.
6: The MultiBRIS server sends the updated route alternatives back to the client.

3.2 Technologies

The aim of MultiBRIS is to reach as many mobile device platforms as possible. This is the key point when choosing a deployment technology, ruling out solutions like Appcelerator Titanium that only supports iOS and Android (As well as solutions placing a heavy burden on the developer to know a particular technology, e.g. Rhodes which requires knowledge of Ruby and has an extensive API). Another prerequisite is that the framework should be easy to use and be able to collaborate with other frameworks that can make it faster and easier to create well-working GUI-components.

This is easier in PhoneGap[5] than in the other deployment technologies. **PhoneGap** is an open-source mobile development framework enabling software programmers to build applications using JavaScript, HTML5 and CSS3. Our MultiBRIS client application does not require heavy computing or graphics, so the performance weakness of PhoneGap is not a big problem. We use PhoneGap as our deployment technology, together with the Sencha Touch JavaScript library for graphical implementation.

The server technology used is Java Servlets, a technology which now is at version 3.0 and has been around for over a decade. With Java Servlets, business logic can be written in Java and then made available to consumers through servlets [Per04]. A Java servlet can be published through any available servlet container, making it very portable. We chose to use the **Jetty**[6] servlet container which is made up of pure Java code, ensuring portability.

4 Results

Migrating the business logic from the phone to the server resulted in two clear benefits, in addition to making the implementation easier: query time reductions were obtained by introducing a new web-interface for BusTUC (Section 4.1), while both the amount of data transfer and the power usage were reduced (Section 4.2).

4.1 Query Time Reduction

The MultiBRIS server has been optimised in two ways. Sharing the ”bus ID to bus-stop live ID” lookup list between all the clients using the server, and by the introduction of threading in the retrieval of real-time data for multiple bus stops. This resulted in computation times twice as fast as the original, as multiple threads can send queries in parallel, instead of sequentially. However, the time from when the query was posed to the server response was still too long, sometimes up to 30 seconds, since the MultiBRIS server had to wait for answers from the BusTUC server. Hence speeding up the BusTUC server was imperative for the practical usability of the entire system.

[5] `http://www.phonegap.com/`
[6] `http://www.eclipse.org/jetty/`

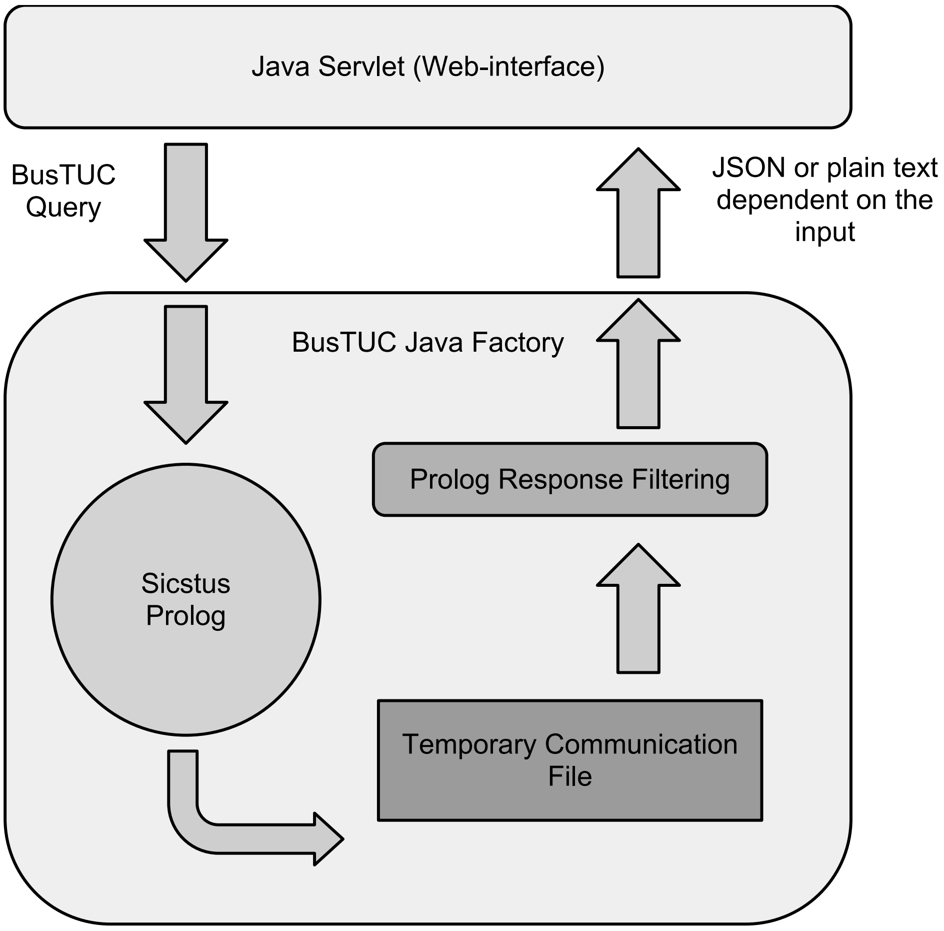

Java Servlet (Web-interface)
BusTUC Query
JSON or plain text dependent on the input
BusTUC Java Factory
Prolog Response Filtering
Sicstus Prolog
Temporary Communication File

Figure 3: The BusTUC web-interface

As a response to this, a new web-interface for BusTUC was built, using Java Servlet technology with the same Jetty container as the MultiBRIS server. Figure 3 shows an overview of the web-interface. When a query arrives at the Java Servlet, the query is sent to what is called the BusTUC Java Factory. The BusTUC factory poses the query to the BusTUC Prolog code, which, in turn, puts the answer into a temporary file. This file is then read and filtered (in the Prolog Response Filtering Module) before the result is returned to the Java Servlet. By creating a new BusTUC web-interface, a substantial reduction in query time was achieved. The average query time was reduced from 15 to 6 seconds, but the most important improvement was in the maximum query time, which was lowered to just one third of the original (from 30 to 10 seconds).

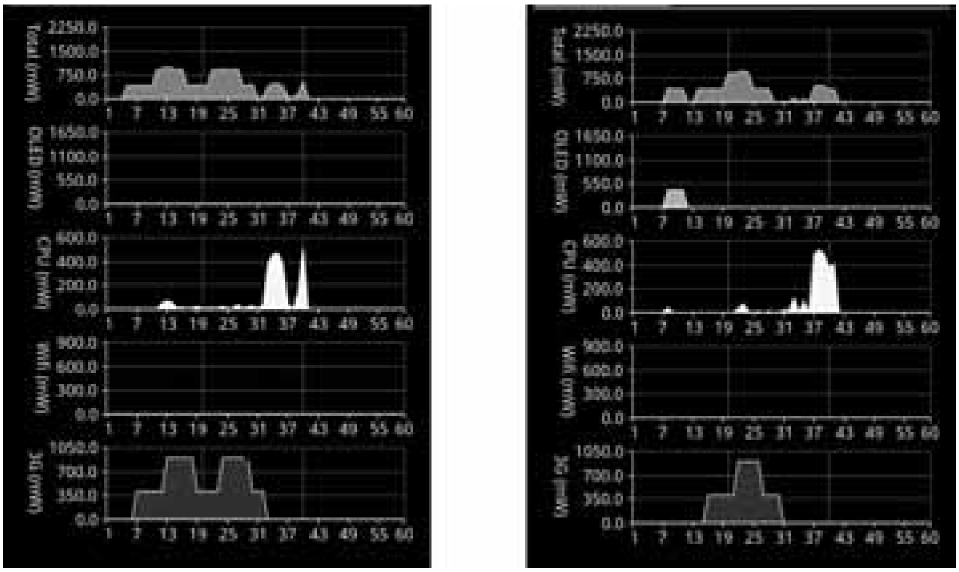
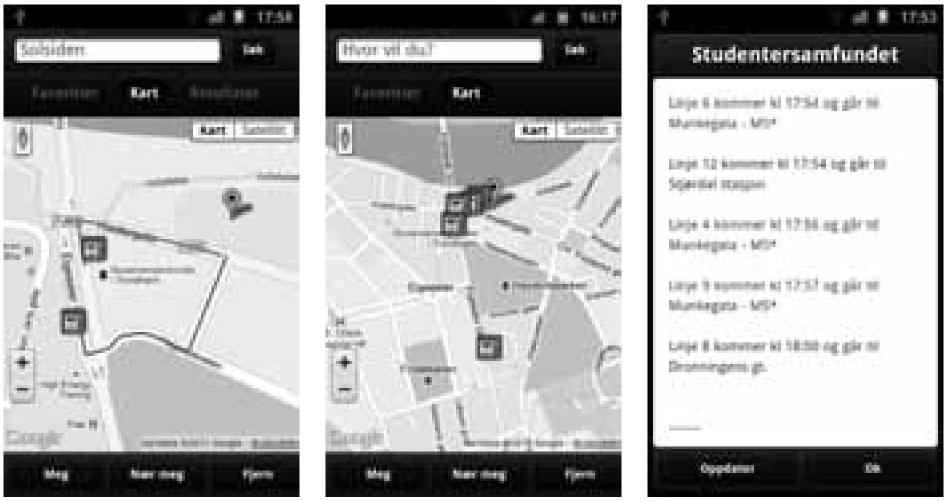


Figure 4: Power usage with business logic on the client (left) vs. on the server

4.2 Server-side Testing

One of the results of moving the business logic to a server is a decrease in the data transfer to the client. If all business logic were to be handled on the client side, all the data would need to be transferred to the client. A lot of data needs to be transferred, since the business logic requires one request to be sent to BusTUC, and several requests to be sent to a SOAP service for the real-time data. By handling the business logic on the server, the server can make all these requests to the various services. Since some of the requested resources can be shared between all the clients using the server, the server can also relieve the external services from heavy request load, by caching the results.

The average data transfer for the client before and after the business logic was moved to the server was reduced from 570 KB to 5 KB (measured with WireShark 1.6.1; the client used was an AppleWebKit 535.2 based browser). The measurement scenario consists of starting the client and making (a bus route alternatives) query, that is, using the "main service" on the MultiBRIS server (Section 3.1). This scenario was chosen as it would represent the most natural usage pattern. About 400 KB of (extra) transferred data comes from downloading the live ID to bus-stop ID list from AtB. Regarding the saved power usage, Figure 4 shows a comparison between power usage for having business logic on the client and on the server. The topmost graphs display the total amount of milliwatts (mW) usage for each application, and clearly show that the solution where the business logic is on the client consumes more power than the server-based solution.



Studentersamfundet

Linje 6 kommer kl 17:54 og går til Munkegata - M5*

Linje 12 kommer kl 17:54 og går til Stjørdal stasjon

Linje 4 kommer kl 17:56 og går til Munkegata - M5*

Linje 9 kommer kl 17:57 og går til Munkegata - M5*

Linje 8 kommer kl 18:00 og går til Dronningens gt.

Oppdater Ok

(a) Results shown on the map (b) Close-by bus stops (c) Real-time information

Figure 5: Screenshots

4.3 The Client Application

When the MultiBRIS client application starts it tries to retrieve the user's current location. A search bar always resides on top of the application, giving the user quick access to bus route search functionality from wherever the user has navigated in the application. The search bar has an auto-complete function that suggests bus stops in Trondheim.

As the result of a search query, the user is presented with a list of the five most optimal bus routes, sorted by total travel time. The user can either tap on the most suitable result to switch to the map and view the target bus stop and travel route to that bus stop, or click on a map tab and be presented with all the bus stops in the result. Coloured lines show the user how to get to the respective bus stops from the current location (Figure 5a).

The map of the application consists of a Google Map with added functionality. The current location is centered on the map when the user taps the "*Meg*" (me) button. In the map tab the user can also click on "*Nær meg*" (close to me) to add nearby bus stops to the map (Figure 5b). The user may click on a bus stop to get real-time information on the next five buses passing through it. Buses are marked in red if they have actual real-time values, or black if only scheduled times are available. If any of the buses are among the user's previous search results, they are also marked with an "*" (Figure 5c).

5 Conclusions and Future Work

The implementation of the MultiBRIS server and update of the BusTUC web-interface proved successful. The system as a whole went from having a client-application that transferred up to 500 kB of data for a bus route query and a query time that was around 20 seconds, to transferring only 5 KB of data and having query times at around 10 seconds. Saving both time and data transfer was imperative for the practical viability of the client. When looking at the power usage in Figure 4, a surprising property was revealed: the difference in power usage in a large part comes from the extra data transferred and not from more CPU usage. Normally, the CPU cycles and the display are portrayed as the main battery power consumers; however, the results here indicate that data transfers can consume as much power as CPU cycles in some cases.

As shown, there are a lot of benefits from moving much of the business logic to a server. The client saves battery and it is easier to maintain and update the system. However, there are some possible drawbacks with adding a server as part of the solution. Doing this effectively creates another layer which the information has to pass through to reach the client, adding another point which can potentially fail to work properly. Another aspect is that the server, when used in a production environment, needs proper infrastructure as a foundation in order to be reliable enough for any client to use. Hence, in a commercial solution, the infrastructure cost for a server-infrastructure has to be considered.

An interesting extension would be to look at systems taking intelligent decisions on where to compute, such as Spectra [FPS02] which dynamical decides whether to perform computation on the server or on the client. "Spectra" monitors resource usage both on server and client, and makes an "optimal choice" based on given system parameters. This functionality could be implemented for MultiBRIS so that if, for example, the MultiBRIS server was under such heavy load that it would delay query times, the clients could be instructed to perform the route calculations and contact the underlying services themselves. However, this would make the client code grow substantially, as it would need to contain all the business logic necessary to perform computations and service calls.

Acknowledgments

Thanks to to LingIT AS and AtB AS for allowing the use of the BusTUC and the real-time bus information services, respectively; to Chistoffer Jun Marcussen and Lars Moland Eliassen for their valuable ideas and feedback during the project; and to Magnus Raaum for providing the source code of the original Android application.

Special thanks to Tore Amble: your memory lives with us and with the BusTUC system which you devoted so much of your life to.

The work of the third author was partly funded by the Research Council of Norway, through the project 187865/S10 UbiCompForAll.[7]

[7]`http://www.sintef.no/Projectweb/UbiCompForAll/`

References

[Amb00] Tore Amble. BusTUC: A Natural Language Bus Route Oracle. In *6th Conference on Applied Natural Language Processing*, pages 1–6, Seattle, Washington, 2000. ACL.

[Chr11] Adam M. Christ. Bridging the Moble App Gap. *Sigma*, 11(1):27–32, October 2011. Special Issue: Inside the Digital Ecosystem.

[FPS02] Jason Flinn, SoYoung Park, and M. Satyanarayanan. Balancing Performance, Energy, and Quality in Pervasive Computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 217–226, Vienna, Austria, July 2002. IEEE.

[FWB10] Brian Ferris, Kari Watkins, and Alan Borning. OneBusAway: Location-Aware Tools for Improving Public Transit Usability. *IEEE Pervasive Computing*, 9(1):13–19, January–March 2010.

[FZ94] George H. Forman and John Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, 27(4):38–47, April 1994.

[Mor09] Daniel Moraff. Google Transit Feed Specification: A Primer. Website, November 2009. `http://www.eot.state.ma.us/downloads/developers_beta/MassDOT_GTFS_Primer.pdf`.

[Pad11] Richard Padley. HTML5 - bridging the mobile platform gap: mobile technologies in scholarly communication. *Serials*, 24(3):S32–S39, November 2011.

[Per04] Bruce W. Perry. *Java Servlet & JSP Cookbook*. O'Reilly Media, Sebastopol, CA, USA, January 2004.

[Pet11] Christy Pettey. Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent. Website, November 2011. `http://www.gartner.com/it/page.jsp?id=1848514`.

[Raa10] Magnus Raaum. An Intelligent Smartphone Application. Master's thesis, Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway, June 2010.

[Vin75] Thaddeus Vincentry. Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations. *Survey Review*, 23(176):88–93, April 1975.

SESSION 3

Health, Wellness and Emergency Support