

BCIT Bachelor of Technology in Computer Systems

Major Project Final Report

COMP 8045 & 8046 (18 Credits)

Submitted as part of the requirements for graduation on December 15, 2008

Copyright © 2008 Stephen Makonin.



Table of Contents

Introduction	3
Background Summaries.....	4
Student Background Summary	4
Educational Background	4
Professional Background	4
Computer Systems Specialization.....	4
BCIT GAIT Lab Background Summary.....	5
Project Management Background Summary.....	5
Alternate Solutions.....	6
Website Topology	6
Visual Map Editing	6
Social Network & Discussion Groups	6
Mobile Device Access.....	6
Chosen Solutions	7
Website Topology	7
Visual Map Editing	7
Social Network & Discussion Groups	7
Mobile Device Access.....	7
Development Details.....	8
Development Methodology.....	8
Analysis & Feature Requirements.....	9
The Mapping System	9
The Social Network.....	10
Mobile Access.....	10
Design & System Architecture	11
System Overview	11
Visual Layout & Layers.....	12
System Layout & Structure	13
Database Design	14
Class Diagrams.....	15
Software Quality Assurance	19
Source Code Control.....	19
Bug Tracking	19
Unit Testing	19
Code Walkthrough.....	19
User Acceptance Testing	19
Configuration Management.....	20
Development Workstations.....	20
Development Web Server	20
Cell Phones	20
Implications of Implementation	21
The Performance of Java.....	21
Server Environments.....	21
Reliance on Facebook and Google Maps	21
Hibernate ORM	22
Turn-By-Turn Directions.....	22
Research in New Technologies	23
Web 2.0	23
Mashups	23
REST	23
GPS Devices	23
Future Enhancements	24
Multi-Destinations	24
Rewrite to PHP.....	24
Multi-Mode Routes.....	24
Conclusions	25
Appendix A: Various Screen Shots.....	26
Appendix B: Letter from Project Sponsor	31

Introduction

This project was an attempt to create and study an online social network that had a common interest. More specifically we wanted to take a segment of the BCIT community, both employees and students, and provide to them a way to share bike routes they use to travel back and forth to BCIT.

The project is called **Map with Wheels (MWW)** which was developed in the BCIT GAIT Lab. It was a TEK funded grassroots project initiative to encourage staff and students to cycle to work and to share their bike routes with others. This project enables sharing through user/community participation. Users can access routes contributed by fellow cyclists in real time via any computer or mobile device (e.g. cell phone).

Map with Wheels utilizes the Facebook social networking software and uses Google Maps to add, edit, and display bike routes and Points of Interest (POI). POIs, such as hazards, controlled intersections, cut-throughs, and other user defined points help describe various situations that may be encountered while taking a particular bike route. Discussion groups further foster communication amongst users as to their experiences while taking a particular bike route. Each route can then be rated in terms of: directness, flatness, control, and traffic volume.

Map with Wheels allowed me to expand my knowledge of Java, Web 2.0, and mobile phone application development. Map with Wheels also will allow me to explore providing solutions using high-level communication methodologies such AJAX and RESTful web services. Exploring Mashups that deals with collaborative computing and social networking (communication at a human level) is also an interest of mine.

There is an online video demonstrating MWW at: <http://server20.ielbcit.ca:81/mww-demo/>.

Background Summaries

Student Background Summary

Educational Background

Stephen Makonin holds a Diploma in Computer Information Systems (1996) from Selkirk College and is in the process of studying for his Bachelor of Technology at BCIT (expecting to graduate in 2010). He along with four other authors wrote the Visual C++ 5 Developer's Guide which Sams Publishing published in 1997.

Professional Background

For the last 12 years, Stephen successfully ran a software development company providing full SDLC services before joining BCIT, in January 2008. In 2007 he, along with another business associate, launched Vvvrroom.com an online news reader and news aggregator which employs a rich user experience built on Web 2.0. His industry knowledge includes: government, healthcare, manufacturing, online services, social networks, and telecom (paging, cellular, wireless). Some of his past business clients include: engcen.com, Fraser Health, Quartech Systems, Sierra Wireless, Telus Mobility, and Vancouver Coastal Health.

Although Stephen's preferred programming languages are C and Python, he has used many others to create and deliver software solutions in: billing/rating, data analysis/conversion, socket programming, web applications, as well as Internet social networking. He has extensive experience using XML (including web services, AJAX, RSS, and ATOM) and implementing Internet RFCs (HTTP, SMTP, POP, NNTP, SNMP, UFTP).

Computer Systems Specialization

Stephen has chosen to specialize in one option which is Data Communications. This specialization allows him to explore his interests in high-level and low-level data communications programming. He believes that data communications is one of the key fundamental and foundational building blocks for technology: past, present, and future.

BCIT GAIT Lab Background Summary

The BCIT GAIT (Group for Advance Information Technology) Lab has a long history at BCIT. It began as the ARCS (Applied Research in Computer Systems) Lab, a CST student research lab, around 1984 headed up by Mike Scrabin. In 1986, headed by Fred Martin, it had structurally developed into what is GAIT today, a research lab that preformed applied research for industry clients. In 1990, the ARCS Lab was renamed to GAIT. GAIT is now headed up by Dr. Hassan Farhangi an Electrical Engineer and the driving force for BCIT research in Smart Micro-Grids.

Through the years, GAIT has focused on AI research then refocused on Internet/network security. At present GAIT has three main research themes: Intelligent Power Grid, Mobile/Wireless Application Development and Web Performance Analysis. Map with Wheels falls under the Mobile/Wireless Application Development theme.

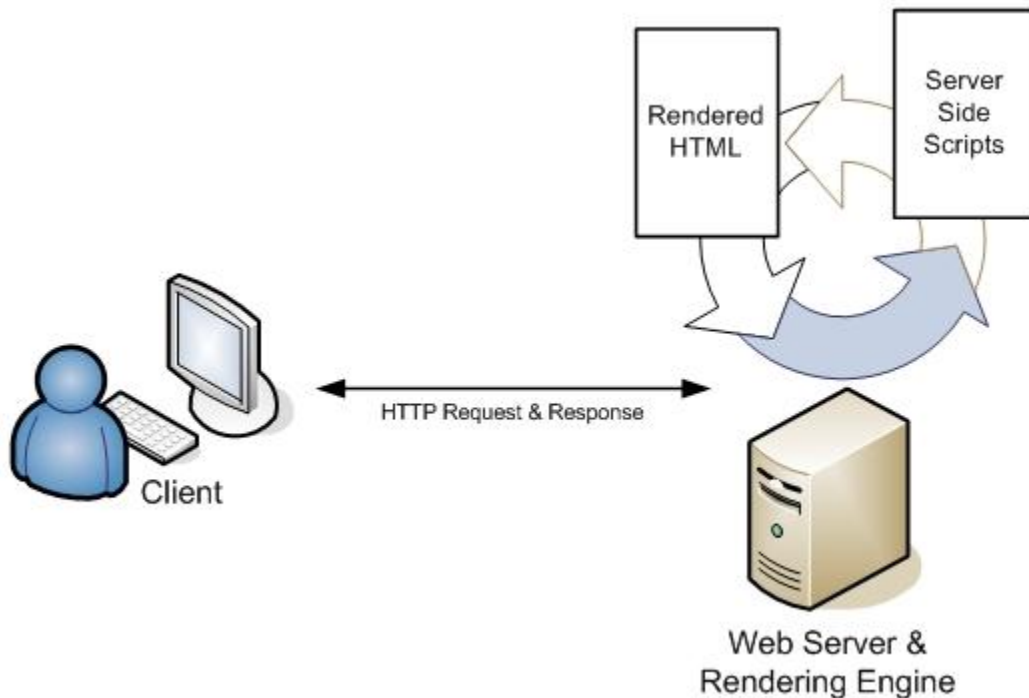
More information about GAIT can be found at: <http://www.bcit.ca/appliedresearch/gait/>.

Project Management Background Summary

Project Management for MWW was performed by Dr. Ari Goelman (TPEG Project Leader) and Clay Howey (GAIT Research Head). MWW was the brainchild of Dr. Ari Goelman, the Primary Investigator for the Social Sciences research that was done. Clay Howey (GAIT Research Head) was Stephen Makonin's project supervisor. Both, Dr. Ari Goelman and Clay Howey, have approved this project for use as my BTech Major Project. Clay Howey has agreed to be the Project Sponsor and his letter is attached under Appendix B.

Alternate Solutions

Website Topology



In traditional web development a page contained server-side code that would render and modify HTML before sending it to the client browser. If a user wanted to view a different webpage or submit a form, the request was made and the server then processes the information and returned a new webpage. This can involve a lot for page refreshing limiting a “good” user experience.

Visual Map Editing

To create a map editing experience is to access and process raw GIS data for companies like Navteq and then create a JavaScript system to display the map data on a webpage. Doing this would require GIS expertise and a significant amount of time to develop the display map system.

Social Network & Discussion Groups

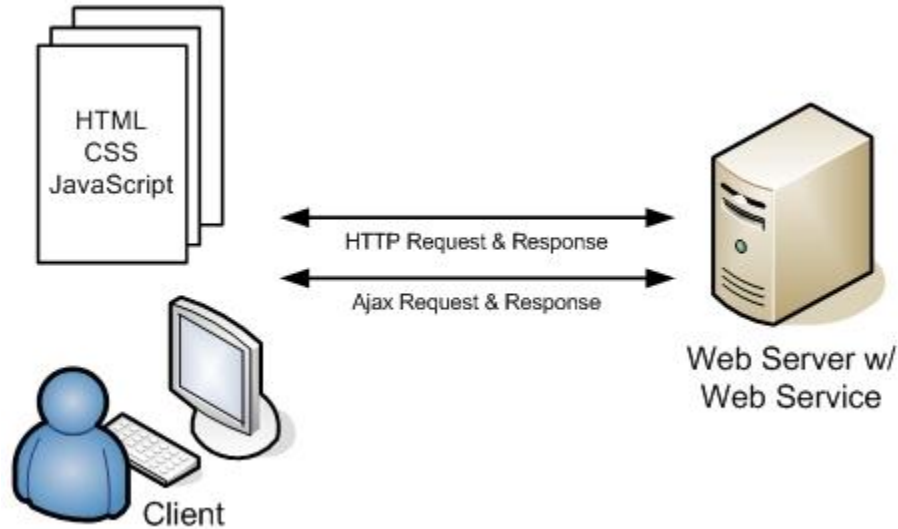
One way to tackle this is to create or use 3rd party discussion group software. Modification would need to be done to allow for more of a social user experience and users would have to learn a new systems.

Mobile Device Access

Create a Symbian application that would interface with GPS devices and then use a micro-browser to display the requested information. This would require that a cell phone user have a micro-browser application on their cell phone and a data plan.

Chosen Solutions

Website Topology



AJAX was used to communicate between the client browser and the server to eliminate page refreshing. JavaScript on the client-side will call RESTful web service URLs which then returns data formatted as JSON results. JavaScript then manipulated the HTML and Google Maps to present the user requests.

Visual Map Editing

Google Maps was chosen to be used for displaying map data. By choosing Google Maps development time was cut down and we were able to provide these more advanced mapping features:

- Moving existing markers on the map and redrawing the route to reflect that.
- Dragging existing markers to new positions on the map.
- Slicing a line into two lines and at the clicked point a new marker is inserted.
- Deleting a marker deletion and redrawing the route to reflect that.
- The opacity of a route segment increases due to its popularity.

Social Network & Discussion Groups

Facebook was chosen to solve the social network and discussion groups issue. Most faculty, staff and students have a Facebook account. Choosing to use Facebook also cut down on the development time.

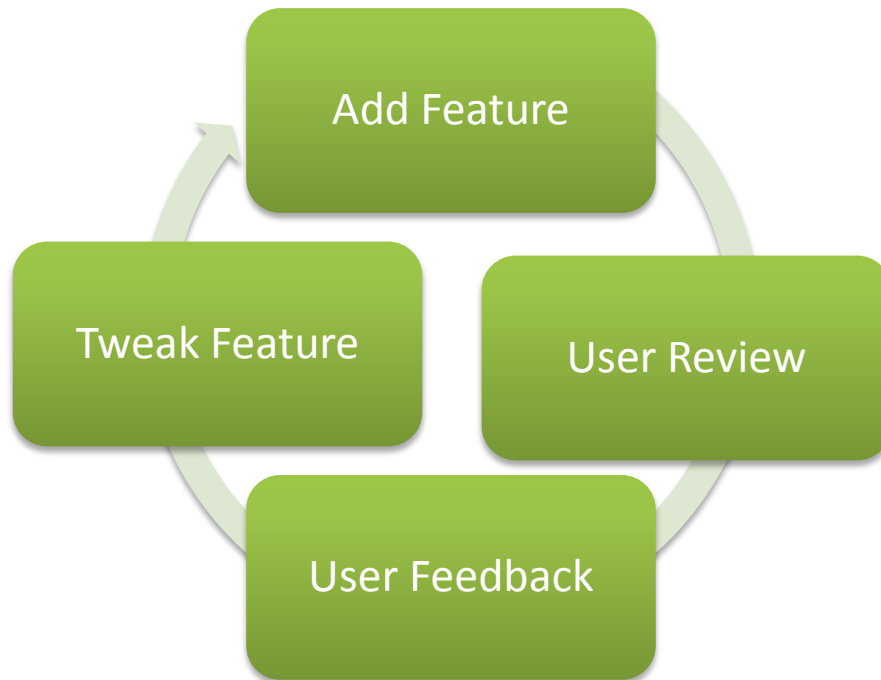
Mobile Device Access

A J2ME application was created instead of a Symbian application. Because J2ME was chosen, the application would be able to run on more cell phones. Bi-directional SMS was used to deliver data to the cell phone. SMS is available on all phones and using the Mobile MUSE Platform to send and receive SMS messages were factors in our decision. Not all cell phone users have a micro-browser application on their cell phones along with a data plan.

Development Details

Development Methodology

Development methodology used was “Design by Prototype”. This methodology allows for a greater degree for end-user interaction and immediate feedback as the project progressed toward completion through each micro-iteration. Micro-iterations are the delivery of one or two useable features in the application that a user can try and test.



As the project is being built the users get to see immediate progress and provide immediate feedback. This methodology is particularly useful for Applied Research when there is no clear vision as to how an end product may look. It also provides for a set of more flexible and dynamic feature set that is customer driven.

Unlike other prototyping development methodologies, an overall plan exists with a preliminary set of features. The overall design and framework are created as a first step. Once this foundation is laid out then features are added and the customer becomes more involved with feedback. The trick, with this methodology, is to create an open architecture that can handle changes to requirements as the project progresses. This methodology would not be recommended for developers with little experience in designing software—senior developers must have that ability to visualize the overall system as well as using the intuition that was developed from participating on many different software projects.

Analysis & Feature Requirements

MWW is comprised of 3 sub-systems: the mapping system, the social network, and mobile access. Scope and related details to each component are listed below:

The Mapping System

Scope

Bike routes will be centric to, and focused on, the BCIT community. It is assumed that the endpoint for a bike route will always be BCIT, Burnaby campus. Users will be able to rate their own routes to help other users find routes that may be more suited for them. Google Maps API will be used to render and display map information and data.

Feature Requirements

The following is a list of specific features sets needed for the mapping system:

1. Routes and POI must be added, edited, and displayed graphically.
2. Routes and POI must also have a name and description.
3. POI must provide the option of uploading a picture for display
4. Routes must be able to be rated by: directness, flatness, control, and traffic volume.
5. Routes must have the option of being visually (colour) displayed relative to a chosen rating.
6. The more popular a section of a route is the darker the route line will get.
7. MWW must have the ability to prompt users to fill out a survey.
8. The mapping interface should be simple and intuitive to use.
9. The map must be user interactive. The user must have the ability to zooming in/out and scroll the map to different places.
10. The map must clearly highlight POIs with intuitive icons.
11. If the mouse hovers over a bike route start marker or POI an information tag is displayed (e.g. "The Brentwood Route by Stephen Makonin").
12. Information windows must have links to allow for editing and viewing discussion forms.
13. Users must be able to delete their own bike routes and POI.

The Social Network

Scope

MWW must foster a community that can use the system to better their experience biking to BCIT. The information they provide will in turn benefit other users. With the success of Facebook, MWW will integrate into it using Facebook's API. This will allow MWW to take advantage of an existing vibrant online social community. Facebook discussion groups will be added to allow users to comment in general or on a specific bike route or POI.

Feature Requirements

The following is a list of specific features sets needed for the social network:

1. Facebook users can search for, add and remove the Facebook App from their account.
2. The main application will be on a Facebook Canvas Page, there must be profile-viewable part allowing other users see and access MWW.
3. If a user attempts to view MWW he/she must be prompted to login if he/she is not.
4. MWW must prompt the user to add the application; if they have not added it yet to their Facebook account.
5. Users can add comments to any discussion group.
6. Users will be able to delete or edit comments once posted.
7. Comments will only support plain text formatting which eliminates complexity and potential rendering issues.

Mobile Access

Scope

Users who have a mobile device that has GPS built to them or is attached to a Bluetooth GPS device can use a downloadable Java mobile application to query MWW. The mobile application will send MWW the GPS coordinates of the user. MWW will then send an SMS that gives directions for the nearest bike route. Only bike routes that end at BCIT will have their directions sent to a requesting mobile phone. The mobile application will be created with the lowest common denominator mobile device in mind. This means that only plain text SMS will be used for communications.

Feature Requirements

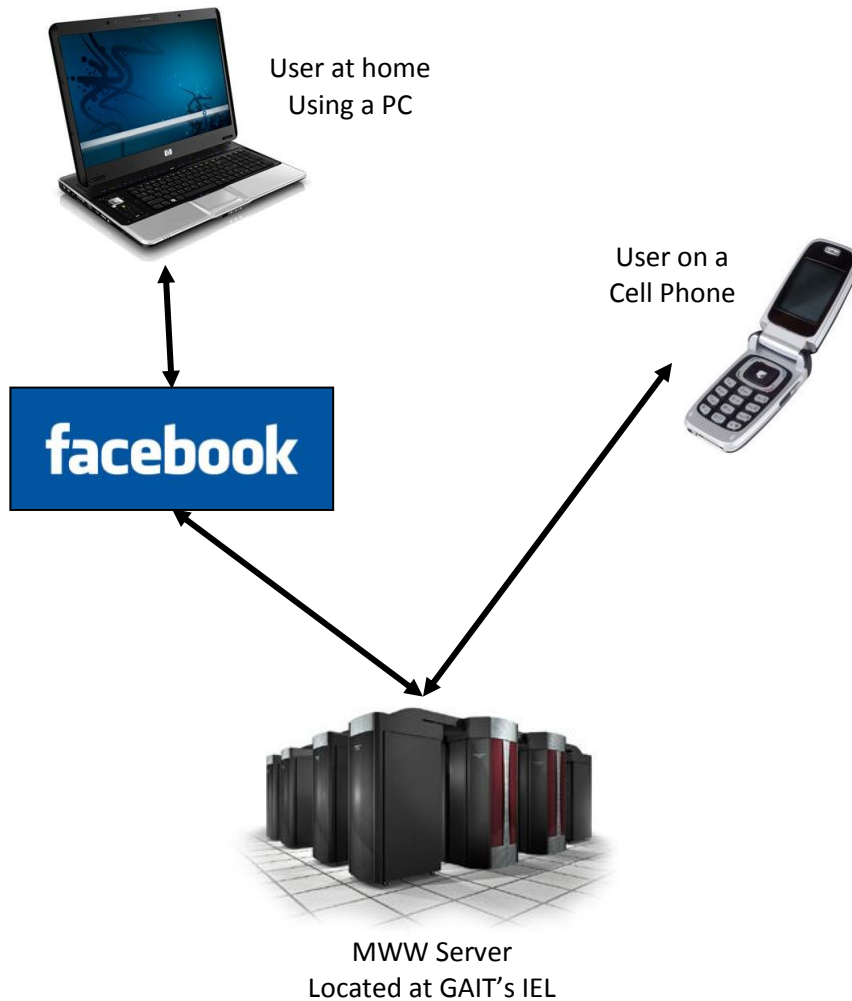
The following is a list of specific features sets needed for mobile access:

1. The mobile application must be able to read and send, via SMS, the user's current coordinates. If the user is not in the Greater Vancouver area an error is displayed.
2. Once MWW receives the user coordinates, an SMS message will be sent back containing turn-by-turn directions of the nearest bike route to BCIT.

Design & System Architecture

System Overview

Users on a computer use MWW through Facebook while users using a cell phone can use a custom Java application to query to the MWW system for nearest bike route. The following is a high-level overview of how the user will connect/communicate with the MWW system:

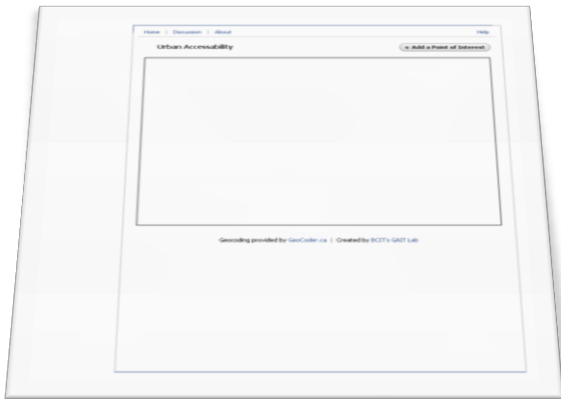


Visual Layout & Layers

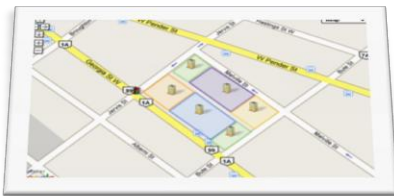
MWW contains a number of visual layers that work in conjunction with each other to provide a rich user experience. Overall layout, in terms of application visual size and webpage position is controlled by Facebook (a.k.a. the Canvas Page). However, within the Canvas Page, MWW has control of all visual and interactive aspects; including such things a layout and page refreshing.



MWW will run within Facebook as an application. Doing so will limit the amount of screen real-estate that can be used. To the right shows the parts of the webpage that are in Facebook’s control. Facebook is considered to be the top of all the visual layers and is the container for all other layers.



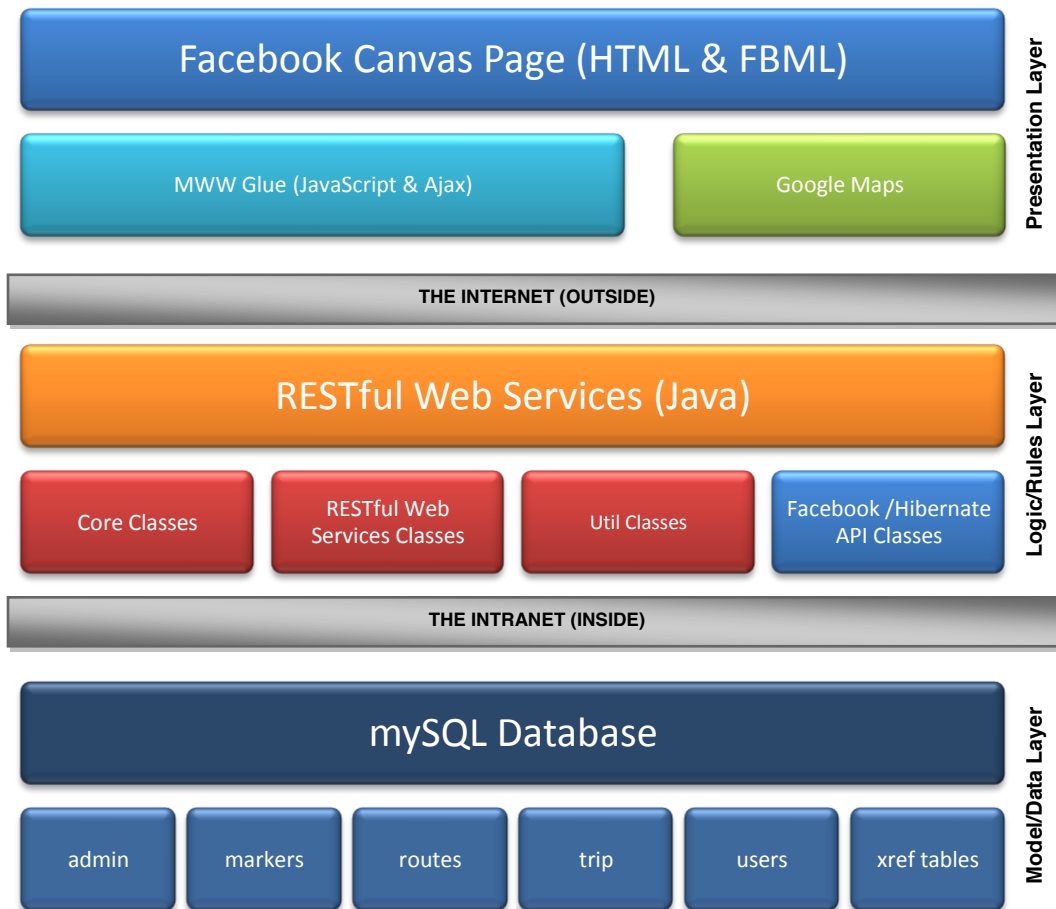
The MWW Glue, the code that controls Google Maps and some of the Facebook integration resides in the middle. To the right shows the parts of the webpage that is the MWW Glue. To limit page refreshes for a better user experience; the MWW Glue uses a combination of JavaScript, Ajax, and Web Services. The result is a look and feel similar to a desktop application.



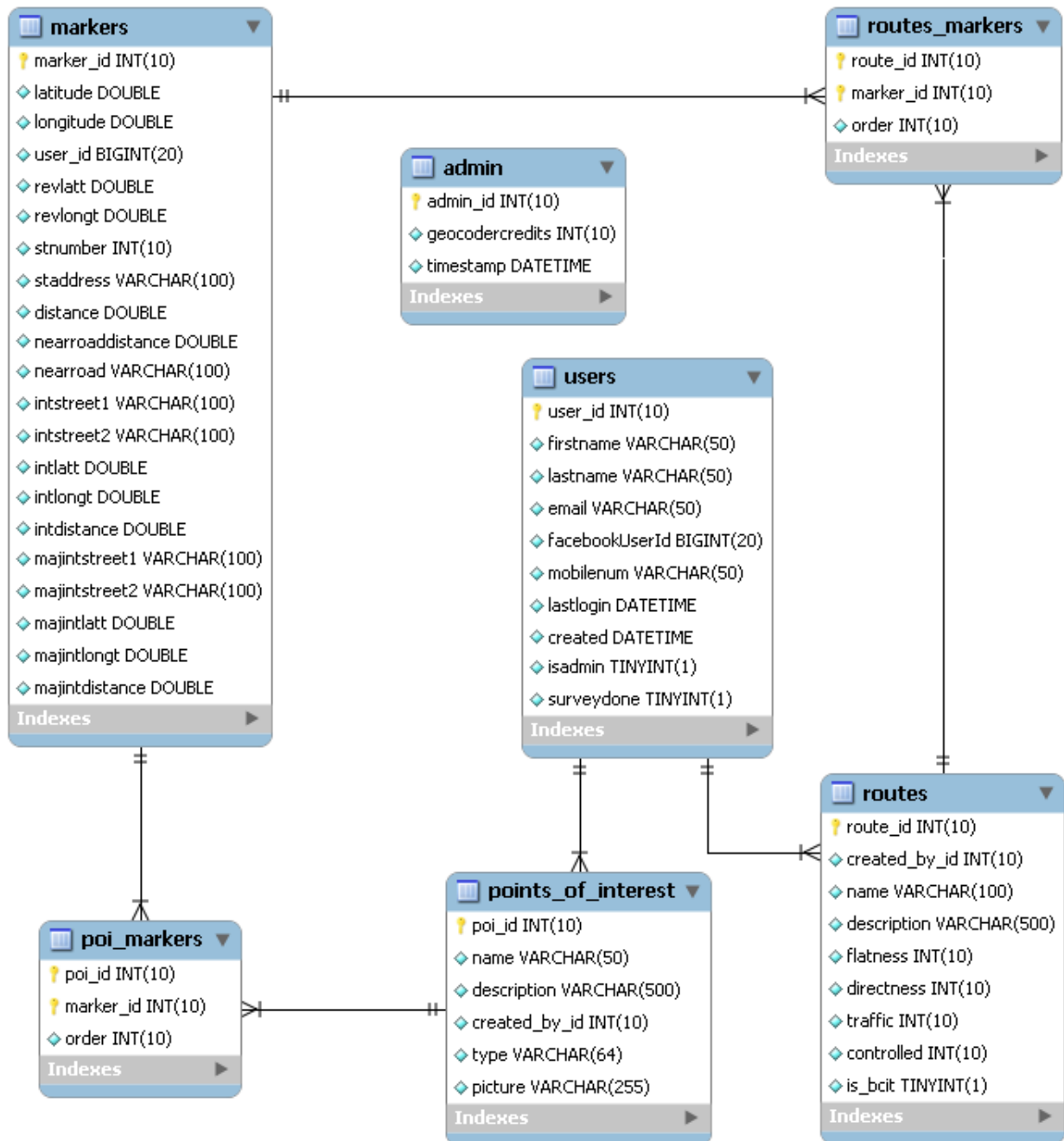
Google Maps is the inner most layer and resides in the inner most box of the MWW Glue. To the right is a sample Google Map.

System Layout & Structure

MWW is considered a multi-layer, or n-tier, application. There is a clear delineation between all layers; physically, technically, and operationally. The Presentation Layer resides within the web browser and is developed using common web scripting languages; mainly HTML, CSS, and JavaScript. The Logic or Rules Layer resides on a web server and is a number of packaged Java classes used to enforce business rules and is a transactional gateway between the Presentation Layer and the Model/Data Layer. The Model or Data Layer resides on a database server (MySQL) and is in fact a database with a number of tables.



Database Design

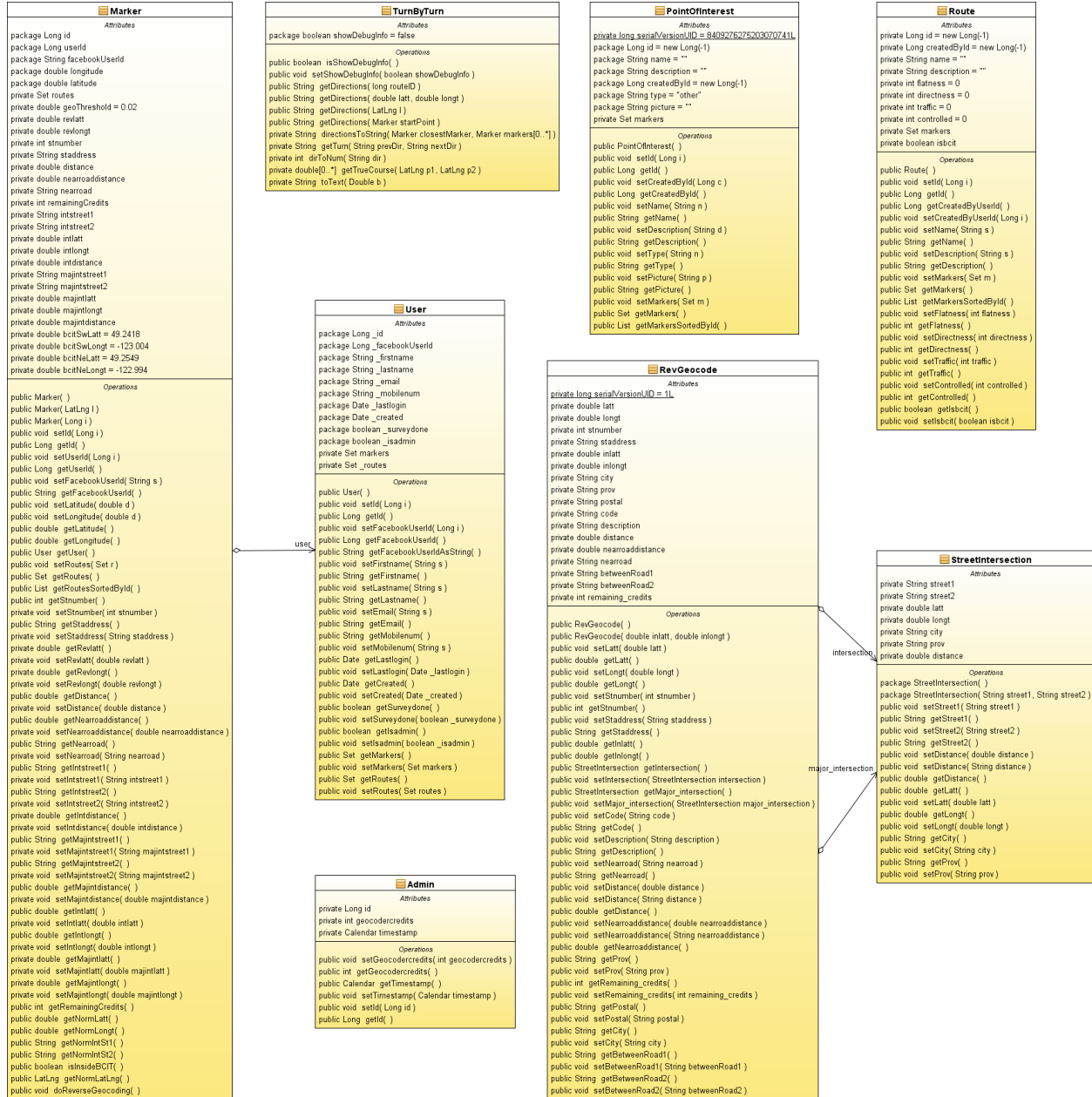


MWW uses a normalized relational table design to store user and system data. The following describes the above tables:

- admin:** stores the counting info for geocoder.ca
- users:** descriptive info on each user subscribed to MWW
- routes:** descriptive info on each bike route (**routes_markers** stores M:N relationship)
- points_of_interest:** descriptive info on each POI (**poi_markers** stores M:N relationship)
- markers:** stores each marker that is used to render a POI or bike route

Class Diagrams

Core Classes



All DB table classes reside in the mww.core package. These classes are responsible for:

- Marker:** Encapsulates the markers DB table; contains data and functions that store and manipulate a marker (or a point on a map).
- TurnByTurn:** Encapsulates code used to create turn-by-turn directions for a list of markers.
- User:** Encapsulates the users DB table; contains data and functions that store and lookup a Facebook account information.
- Admin:** Encapsulates the admin DB table; used to store information on number of time Geocoder.ca was called and with it was last called.
- PointOfInterest:** Encapsulates the points_of_interest DB table (including poi_markers); contains data and functions that store and manipulate a point of interest (POI, or special/flagged marker).
- RevGeocode:** Encapsulates the use of the utility classes that call geocoder.ca.
- Route:** Encapsulates the routes DB table (including route_markers); contains data and functions that store and manipulate a route (series of marker that go from point A to point B).
- StreetIntersection:** Encapsulates the location data from a reverse geocode.

3rd Party API Classes

```

Application
Attributes
private long serialVersionUID = -541404706087860808L
private String fbApplicationApiKey = "ad16a092dbf343a5bf649ad0f9c3c25f"
private String fbApplicationSecret = "129f0a0115d38aa10a0a66e68120a0e8"

Operations
public FacebookXmlRestClient getClient( HttpServletRequest request )
public void doLogin( HttpServletResponse response )
public void doAddApp( HttpServletResponse response )
public String getFbUserID( HttpServletRequest request )
    
```

```

CallbackProcessor
Attributes

Operations
public CallbackProcessor( )
protected void doGet( HttpServletRequest request, HttpServletResponse response )
protected void doPost( HttpServletRequest request, HttpServletResponse response )
    
```

```

HibernateUtil
Attributes
private SessionFactory sessionFactory

Operations
public SessionFactory getSessionFactory( )
    
```

All 3rd party API classes reside in either the mww.facebook or mww.hibernate package. These classes are responsible for:

- Application:** Contains Facebook authentication info and high-level helper functions
- CallbackProcessor:** Responsible for processing calls from Facebook
- HibernateUtil:** High-level helper functions for the Hibernate API

RESTful Web Services Classes

```

Localizer
Attributes
private long serialVersionUID = 724937808708482528L

Operations
public Localizer( )
protected void doPost( HttpServletRequest request, HttpServletResponse response )
protected void doGet( HttpServletRequest request, HttpServletResponse response )
    
```

```

Routes
Attributes
private long serialVersionUID = 724937808708482528L

Operations
public Routes( )
protected void doPost( HttpServletRequest request, HttpServletResponse response )
protected void doDelete( HttpServletRequest request, HttpServletResponse response )
protected void doGet( HttpServletRequest request, HttpServletResponse response )
    
```

```

Picture
Attributes
private long serialVersionUID = 724937808708482528L

Operations
public Picture( )
protected void doPost( HttpServletRequest request, HttpServletResponse response )
private void copyFile( File in, File out )
protected void doGet( HttpServletRequest request, HttpServletResponse response )
    
```

```

ThankYou
Attributes
private long serialVersionUID = 724937808708482528L

Operations
public ThankYou( )
protected void doPost( HttpServletRequest request, HttpServletResponse response )
protected void doGet( HttpServletRequest request, HttpServletResponse response )
protected void doDelete( HttpServletRequest request, HttpServletResponse response )
    
```

```

PointsOfInterest
Attributes
private long serialVersionUID = 724937808708482528L

Operations
public PointsOfInterest( )
protected void doPost( HttpServletRequest request, HttpServletResponse response )
protected void doDelete( HttpServletRequest request, HttpServletResponse response )
protected void doGet( HttpServletRequest request, HttpServletResponse response )
    
```

All servlets reside in either the mww.restful or mww.survey package. These classes are responsible for:

- Localizer:** Serves XML out to the Mobile MUSE Platform
- Picture:** Serves uploading and downloading of pictures
- PointsOfInterest:** Serves JSON POI data out and add/edit/delete POIs
- Routes:** Serves JSON route data out and add/edit/delete routes
- ThankYou:** Serves catching calls from the survey website

Utility Classes

```

GeoCoderHandler
    Attributes
package Boolean inGeodata = false
package Boolean inIntersection = false
package Boolean inMajor_intersection = false
package Boolean inError = false
package Boolean inLatt = false
package Boolean inLongt = false
package Boolean inStnumber = false
package Boolean inStaddress = false
package Boolean inNearroad = false
package Boolean inNearroaddistance = false
package Boolean inPostal = false
package Boolean inRemaining_credits = false
package Boolean inBetweenRoad1 = false
package Boolean inBetweenRoad2 = false
package Boolean inDistance = false
package Boolean inCity = false
package Boolean inProv = false
package Boolean inStreet = false
package Boolean inStreet2 = false
package Boolean inLattx = false
package Boolean inLongtx = false
package Boolean inCode = false
package Boolean inDescription = false
package int level = 0

    Operations
public GeoCoderHandler( RevGeocode out )
public void characters( char ch[0..*], int start, int length )
public void endDocument( )
public void endElement( String namespaceURI, String localName, String qName )
public void endPrefixMapping( String prefix )
public void ignorableWhitespace( char ch[0..*], int start, int length )
public void processingInstruction( String target, String data )
public void setDocumentLocator( Locator locator )
public void skippedEntity( String name )
public void startDocument( )
public void startElement( String namespaceURI, String localName, String qName, Attributes atts )
public void startPrefixMapping( String prefix, String uri )
    
```

```

GeoCoderAdapter
    Attributes
package String AUTHCODE = "650517694850108725794x581"
package String URL = "http://geocoder.ca/"
private double reqLatt
private double reqLongt

    Operations
public RevGeocode getResult( )
public GeoCoderAdapter( double latt, double longt )
    
```

```

MarkerIdComparator
    Attributes

    Operations
public MarkerIdComparator( )
public int compare( Object marker, Object anotherMarker )
    
```

```

RouteldComparator
    Attributes

    Operations
public RouteldComparator( )
public int compare( Object route, Object anotherRoute )
    
```

All utility classes reside in the mww.util package. These classes are responsible for:

- GeoCoderHandler:** Responsible for connecting and communicating to geocoder.ca
- GeoCoderAdapter:** Contains the data for Reverse Goe Coding
- MarkerIdComparator:** The comparator for Marker classes ID
- RouteldComparator:** The comparator for Route classes ID

Software Quality Assurance

Source Code Control

Subversion (SVN) was used to track and version all source code changes. The SVN repository resided on a remote server. SVN clients such as TortoiseSVN for windows and SVN Workbench for Linux were used on development workstations.

Bug Tracking

Mantis was used to manage bugs and track issues during the testing of the Beta release and the production release. Mantis was accessed via a web browser and installed on a remote web server.

Unit Testing

Informal unit testing was done on each code block (e.g. function, class) during and at the end of the coding of each code block. Because of lack of debugging facilities in JavaScript and the need for UI interaction, all unit testing was done manually.

Code Walkthrough

After a significant portion of the Project was developed a Code Walkthrough was done to review architecture, project layout, and overall code quality. The code walkthrough was done informally.

User Acceptance Testing

UAT was done on a daily basis by the Project Manager and by the GAIT Lab. During June to August a Beta Testing Group was created, consisting of about 12 users from around the BCIT Burnaby Campus to use MWW and provide feedback. All bugs were recorded in Mantis.

Configuration Management

Development Workstations

Development workstation was configured as follows:

Operating System:	Ubuntu 7.10 (Gusty Gibbon) or greater
Development IDE:	Eclipse Europa
Java Environment:	J2SE 1.6, ANT, Tomcat 5.5
Object Relational Modeler:	Hibernate
Database IDE:	MySQL Administrator
SVN Interface:	SVN Workbench

ANT build scripts were created to compile the project then copy the .war file to the development server for update.

Development Web Server

The web server had the following configuration:

Operating System:	CentOS 4.6 or greater
Web Server:	Tomcat 5.5
Database Server:	MySQL 5 Latest Version
Other Daemons:	SSH w/ SFTP

This configuration would also be necessary for the production server.

Cell Phones

In order to use MWW on a mobile device a custom application must be downloaded and installed on a user's cell phone. The cell phone must have the following capabilities:

- Bluetooth Enabled (for Holux GPS Device) or Built-In GPS
- SMS Enabled
- J2ME Enabled

Directions to the nearest bike route were sent via Bi-Directional SMS using the Mobile MUSE platform.

Implications of Implementation

The Performance of Java

To compile Java and create .war files, Tomcat needed to be running on the workstation for the each compile. There were issues with Eclipse IDE responsiveness. ANT build times were often slow causing long compile times. Issues such as these contributed to speed of development and caused programmer frustration.

On the mobile side, the challenge will be to create an intuitive mobile device application that will run on the lowest common denominator device. In J2ME, the Location API is not included on a majority of mobile devices. This means that a library will need to be created that can talk to Bluetooth and GPS devices. The library will then need to read and parse GPS data to get the latitude and longitude for the GPS device.

Server Environments

There were some unexpected server issues. There is a bug in the Java VM that prevented the use of AMD processor servers to be used. If an AMD processor was used the Tomcat web server would not responded correctly to Facebook causing Facebook to report an error. Each server needs its own Google Maps API Key. Facebook must also be configured to point to the server where MWW will be running from.

The performance of Tomcat was disappointing. Tomcat seems to have a slow response time and can only handle a load of 10 or so users. In fact, pressing the F5 (refresh) key rapidly in Facebook would cause the Tomcat server to crash and become non-responsive. To ensure some reliability, scripts were created to reboot the server once a night.

Every time the .war file was uncompressed, the current web directory was deleted and all uploaded images would be deleted as well. Extra code was created to store the images in a second place for safe keeping and then copied to the current running web directory.

Reliance on Facebook and Google Maps

Relying on 3rd party sites, as Mashups often do, creates a dependence on that Sites performance and changes. Various issues came from this reliance. The Facebook API is famous for its lack of documentation, so a lot of experimentation was done to get MWW to work in Facebook; some of the issues encountered included JavaScript execution and support limitations. Java API documentation and library code was also out of sync. The official Java API was then dropped by Facebook and Facebook made no guaranties that future changes to its API would be backwards compatible.

There were refresh and performance issues due to inherent problems with Facebook's architecture and network topology. Applications updates could take 2-3 minutes to propagate through to the entire web caching servers so rapid debugging became an issue. As well, the use of the Facebook API coupled MWW too closely to Facebook. MWW could not be run in a standalone version without major programming concessions.

Hibernate ORM

The use of Hibernate added development time to the project so much that it would have been faster to create our own Data Access Layer. The lack of documentation and lack of ORM tools (e.g. automated DAL code generation such as the ones provided by LLBLGen) for Hibernate cause confusion and forced the database design to change. Issues getting one-to-many relationships to work caused redesign and reprogramming of these table relationship to be many-to-many; making the database more complex than it needed to be.

Turn-By-Turn Directions

The challenge will be to take a series of latitude and longitude coordinates and turn them into human readable directions. For example:

```
{ "latt": 49.269225, "longt": -122.992453 },
{ "latt": 49.269288, "longt": -122.996742 },
{ "latt": 49.267453, "longt": -122.997814 },
{ "latt": 49.264413, "longt": -122.997824 },
{ "latt": 49.264408, "longt": -123.00337 },
{ "latt": 49.2612513420326, "longt": -123.003444671631 },
{ "latt": 49.2592347950603, "longt": -123.004302978516 },
{ "latt": 49.254869, "longt": -123.004329 },
{ "latt": 49.25188, "longt": -123.00441 },
{ "latt": 49.2517560473552, "longt": -123.002371788025 }
```



```
Your closest to the Brentlawn route. Directions:
Start at 1706 Delta AVE near the Highlawn Dr intersection.
Head W on Highlawn DR (0.31 km).
Turn Left at Beta Ave (0.56 km).
Turn Right at Dawson St (0.40 km).
Turn Left at Willingdon Ave (1.40 km).
Turn Left at Goard Way (0.15 km).
You're now at BCIT (total 2.82 km).
```

One of the biggest issues was the lack of GIS expertise needed to understand Great Circle Math algorithms. Example code was eventually found on the Internet and converted into a Java function. Without finding the example code, a cruder version of the above directions would have been delivered.

Research in New Technologies

Web 2.0

The use of asynchronous HTTP request (Ajax) was used to communicate between the client browser and the server to eliminate page refreshing and to provide an enhanced user experience similar to a desktop application. The use of Ajax is a new emerging technology which requires a lot of manual tweaking and debugging to perform correctly.

Mashups

A Mashup involves creating a website with data and other elements from a number of other website APIs and/or other website data aggregators. This can inherently cause issues due to the reliance of other websites/sources to be stable and readily available for use. If one of the websites/sources goes down or responds slowly then it has an immediate and adverse negative impact to your website.

REST

For communication between client browser and server, RESTful web services were used. RESTful web services are a new emerging technology that is an alternative to SOAP. SOAP is complicated and a lot of overhead is needed when using JavaScript and Ajax. RESTful web services use all of the HTTP protocol (GET, POST, PUT, and DELETE) unlike SOAP which only uses POST. Because of this, RESTful web services are resource oriented (or noun-based) unlike SOAP which is more verb-based in design.

GPS Devices

There was a need to create a library/framework that will allow a mobile device to communicate with a Bluetooth enabled PSP device. With the absence of the J2ME Location API on the majority of mobile devices, ways of creating a Bluetooth communication API needed to be researched.

Future Enhancements

Multi-Destinations

MWW can grow and improve in many ways. One such way would be to allow MWW to have multiple destination points, not just BCIT. This would allow a greater user base and would not take much effort to implement. A table would need to be added to store the different destinations points. A user profile screen would also need to be added to allow a user to select and store their default destination point. Only POIs or bike routes for the users selected destination would be rendered. This would be done to speed up the rendering of the map for improved response times and better user experience. JavaScript is often implemented poorly in browsers and functions that involve graphical rendering are very inefficient causing noticeable slowdowns.

Rewrite to PHP

In September of 2008, Facebook dropped support for their Java API. The only official API Facebook now supports is PHP. There are 3rd party Java APIs but they are slow to keep up and have their own set of issues and bugs. Going forward, a rewrite should be done to move from Java to PHP. With traditional web development such a rewrite would be a big effort; not so with Ajax and web services. The majority of the MWW code base is in HTML, CSS, and JavaScript. Only the web services portion of the project was written in Java and only that needs to be rewritten. The web services code portion of the project amounts to about 30% of the code base.

Multi-Mode Routes

Another idea to enhance MWW would be to integrate public transit, and/or bike rental stations, and/or hitch-a-ride stops. These new features would expand the usage of MWW creating a community of users that take alternate forms of transportation. This would also allow people to create “multi-mode” routes. Adding these new features would be risky and a more time consuming task than the above two enhancements. Serious thought to modifications to the UI and database would need to be done to implement this new functionality.

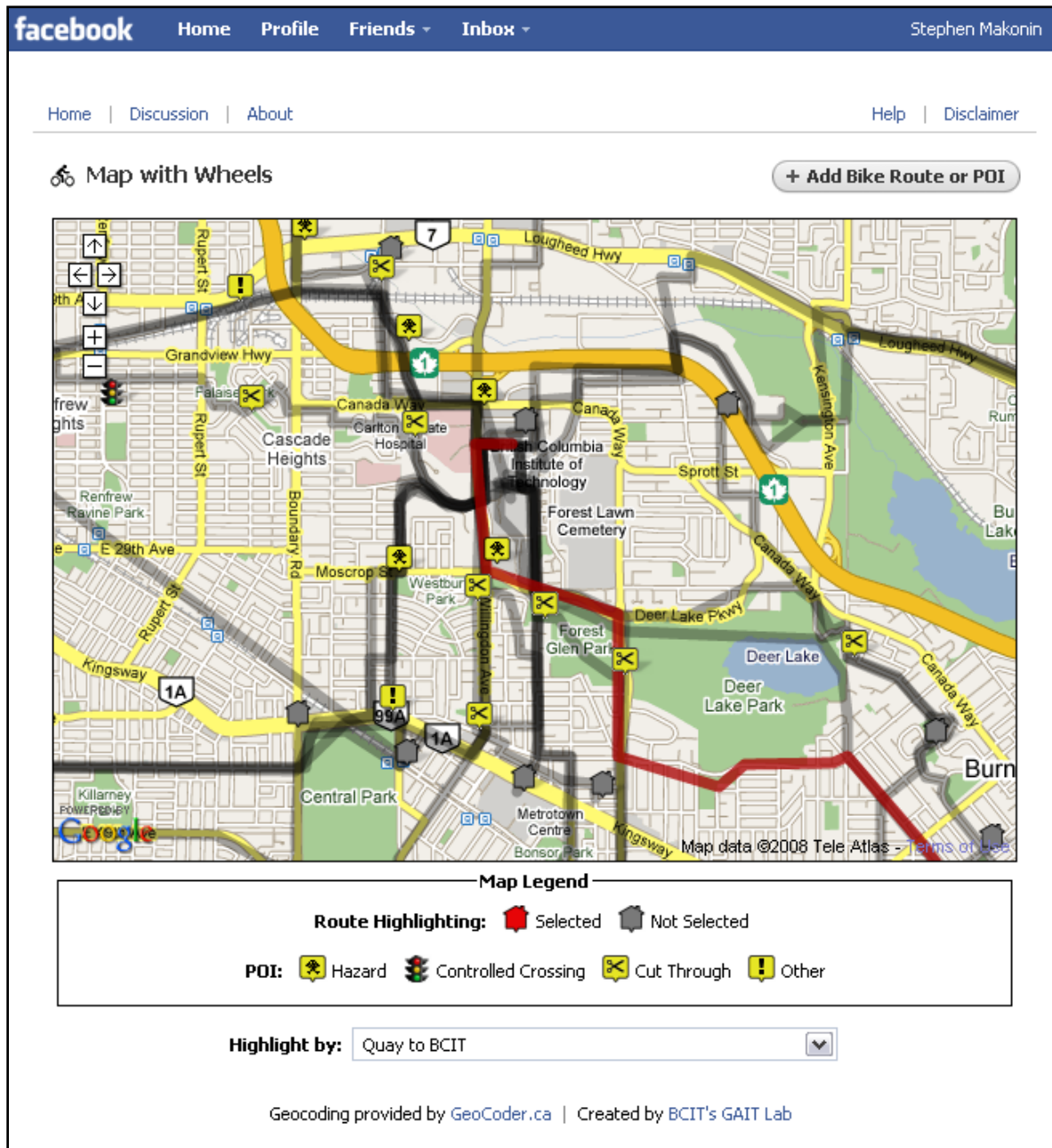
Conclusions

Web services are increasingly becoming a more widely used solution to connect heterogeneous systems. Long gone are the days where projects were created to replace viable legacy systems. By using a simple protocol like HTTP, software that exists on these heterogeneous systems can be exposed and ran by any other system. In the end, it is much cheaper and less risky to add a web server to a legacy system then it is to rewrite it.

The uses of social networks, such as Facebook, are becoming a normal part of the everyday life of individuals. By writing an application for a social network you have instant access to these users, developers worry less about Google site placement and creating a standalone system that users have to signup to use. By writing an application for a social network, the adoption rate for application use is much higher as it works on the principle of “word of mouth”.

The Map with Wheels project provided an excellent opportunity to use Java and RESTful web services. It allowed me to explore the concept and the challenges of creating a Mashup. It also allowed me to work on a project that was research based which is often different for the business-driven type projects I normally work on.

Appendix A: Various Screen Shots



This the main page of MWW.

facebook Home Profile Friends ▾ Inbox ▾ Stephen Makonin

Home | Discussion | About Help | Disclaimer

🚲 Map with Wheels + Add Bike Route or POI

Brentlawn by Stephen Makonin

A short but dangerous route!

Take this route if you're late. Cycling it will make you say, "These hills aren't so bad." Crossing major roads is easy - lots of controlled intersections. It has you playing in lots of traffic.

[Highlight](#) | [Edit](#) | [Discussion](#) | [Delete](#)

Map Legend

Route Highlighting: Selected Not Selected

POI: Hazard Controlled Crossing Cut Through Other

Highlight by: Brentlawn

Geocoding provided by [GeoCoder.ca](#) | Created by [BCIT's GAIT Lab](#)

This is an example of an information window.

The screenshot shows the Facebook 'Map with Wheels' interface. At the top, there are navigation links for Home, Profile, Friends, and Inbox, along with the user's name 'Stephen Makonin'. Below this, there are links for Home, Discussion, About, Help, and Disclaimer. The main heading is 'Map with Wheels' with a '+ Add Bike Route or POI' button. Below the heading are two buttons: 'Add Bike Route' and 'Add POI (Point of Interest)'. The central part of the interface is a map of Vancouver, British Columbia, showing a yellow bike route. The route starts at the intersection of E Hastings St and Boundary Rd, goes south on Boundary Rd, then east on Lougheed Hwy, then south on Kensington Ave, and finally east on Lougheed Hwy. There are three red pins on the map indicating key points along the route. Below the map, there is a red instruction: 'Find your starting point. Click on intersections that change your route heading.' Below the map, there is a text input field for the route name, which contains 'My Route'. Below that is a text area for a description, which contains 'This is the route I take everyday... rain or shine!'. Below the description, there are four dropdown menus for rating the route: 'How direct is your route?' (selected: 'Take this route if you're late'), 'How flat is your route?' (selected: 'These hills aren't so bad.'), 'Are there controlled intersections?' (selected: 'Easy - lots of controlled intersections'), and 'How much car traffic?' (selected: 'Has you bike in some car traffic'). At the bottom, there is a red warning: 'Please only add legitimate/real bike routes. All other will be deleted by the the Admins.' and two buttons: 'Save' and 'or Cancel'.

This is the form used to add/edit bike routes.

facebook Home Profile Friends Inbox Stephen Makonin

Home Discussion About Help Disclaimer

Map with Wheels + Add Bike Route or POI

Add Bike Route Add POI (Point of Interest)

Find and click on the location you want to add a POI to.

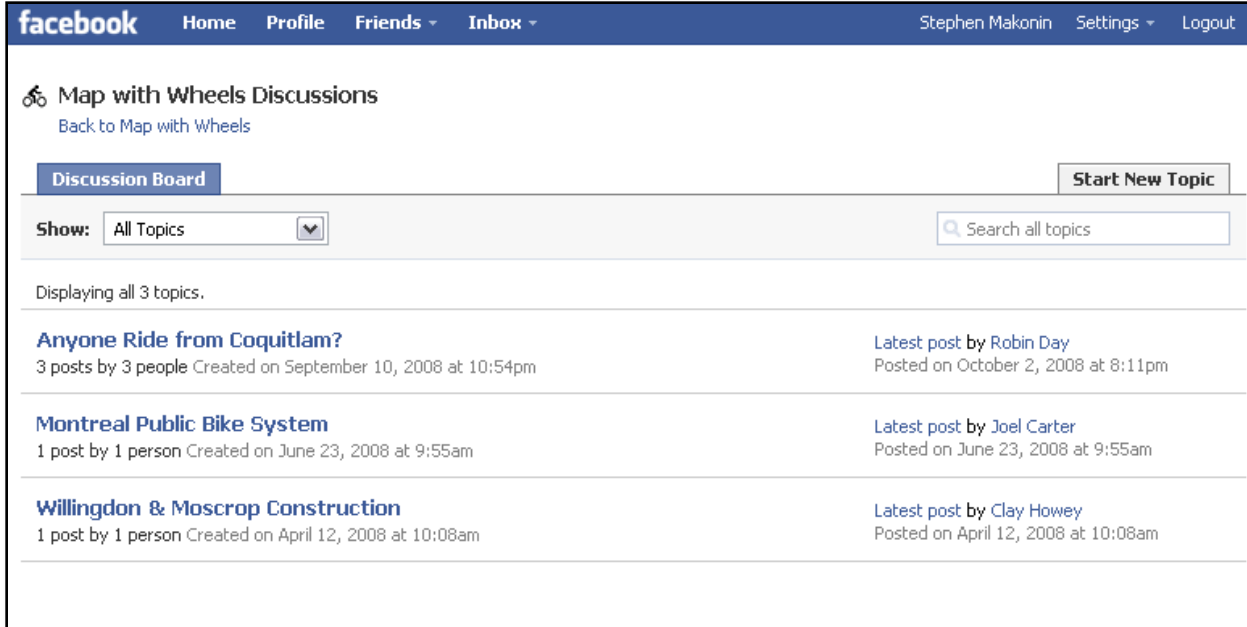
Select the type of POI you would like to create. Select a route to be drawn on the map so you can place a POI on it easily. Describe your POI in more detail. You can, optionally, add a picture for others to see.

Type of Map POI? Pick one...
Show Route: Pick one...
Name Your POI:
Describe It:
Add a Picture (1MB Limit): Choose File No file chosen

Please only add legitimate/real POI. All other will be deleted by the the Admins.

Save or Cancel

This is the form used to add/edit POI.



The screenshot shows the Facebook interface for a discussion board. At the top, the Facebook logo is on the left, and navigation links for Home, Profile, Friends, and Inbox are in the center. On the right, the user's name 'Stephen Makonin' and links for Settings and Logout are visible. Below the navigation bar, the page title is 'Map with Wheels Discussions' with a bicycle icon and a 'Back to Map with Wheels' link. A 'Discussion Board' tab is active, and a 'Start New Topic' button is on the right. A 'Show:' dropdown menu is set to 'All Topics', and a search box contains the text 'Search all topics'. Below this, it says 'Displaying all 3 topics.' Three discussion topics are listed:

Topic Name	Posts	Created	Latest Post By	Posted
Anyone Ride from Coquitlam?	3 posts by 3 people	Created on September 10, 2008 at 10:54pm	Robin Day	Posted on October 2, 2008 at 8:11pm
Montreal Public Bike System	1 post by 1 person	Created on June 23, 2008 at 9:55am	Joel Carter	Posted on June 23, 2008 at 9:55am
Willingdon & Moscrop Construction	1 post by 1 person	Created on April 12, 2008 at 10:08am	Clay Howey	Posted on April 12, 2008 at 10:08am

This is an example of what the discussion groups look like.

Appendix B: Letter from Project Sponsor

December 15, 2008

To the Practicum Review Committee;

I am writing you to confirm that Stephen Makonin worked full-time on the Map with Wheels project. Stephen Makonin worked on Map with Wheels full-time from January 26th, 2008 to May 1st, 2008 and part-time until August 8th, 2008. The project was completed, released, and is currently running in production. The Facebook application is being used by the BCIT community and a number of people have contributed bike routes and points of interest.

Sincerely,

Clay Howey
Research Head
BCIT GAIT Lab

BCIT Bachelor of Technology in Computer Systems

Major Project Supplement

COMP 8045 & 8046 (18 Credits)

Copyright © 2008 Stephen Makonin.



Table of Contents

Introduction	3
System & Library API Interaction	4
Turn-By-Turn Directions.....	5
Ajax & RESTful Web Services	5
Bike Route Editing.....	9
J2ME & GPS Devices.....	10

Introduction

This document is intended to accompany the CST BTech Major Project Final Report document; as it is an extension of that document. The goal is to further satisfy the curiosity of technically-minded individuals who have read the Final Report and would like to see specific implementation details. The **Map with Wheels (MWW)** project has covered a number of specific areas that can be expanded upon.

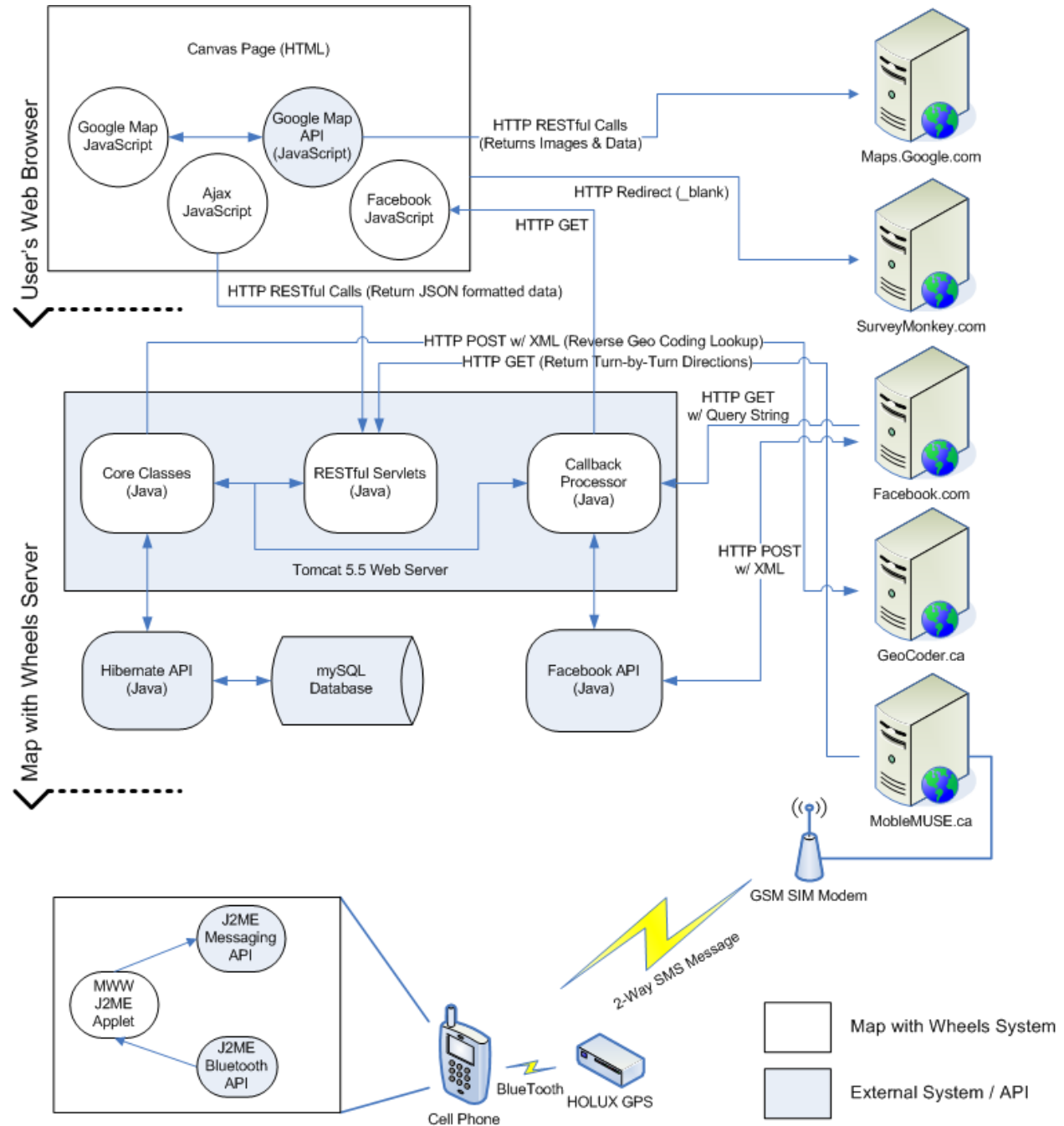
A diagram details the interaction that the Map with Wheels codebase has with external libraries, APIs, services, and systems is presented with communication details between these modules. Implementation details and snippets are provided to further explain how issues were tackled in the following areas:

- Turn-By-Turn Directions
- Ajax & RESTful Web Services
- Bike Route Editing
- J2ME & GPS Devices

Please refer to specific areas within the Final Report to gain a high-level understanding and overview of each subject expanded upon in this Supplement.

System & Library API Interaction

Map with Wheels used a number of external libraries, APIs, services, and systems to provide its full feature set. The following diagram illustrates the relationship and communications used between Map with Wheels and all other external entities:



Turn-By-Turn Directions

This area of the code presented some challenges. At first I did not even know what to call this area for GIS to be able to do proper searches. After about one full day of Google searching I found the subject matter I was looking for. This was an area of GIS mathematics called Great Circle. One of the issues with the generating directions was the calculation of Bearing, which changes when travelling from point A to point B. There are also different results in finding North, South, East and West when calculating Initial Bearing vs. Final Bearing.

Further searching lead me to understand that doing the calculation of True Course would give me the results I desired. I found a Java Applet that does this very thing (<http://lionel.le.taltec.free.fr/2/TH/calc/>). Searching on Google for True Course let me to <http://williams.best.vwh.net/gccalc.htm>; code written in JavaScript to do this. I was able to convert the code to Java as seen below:

```
info = getTrueCourse(m.getNormLatLng(), n.getNormLatLng());
currentDistance = info[1];
currentDir = toText(info[0]);

private double[] getTrueCourse(LatLng p1, LatLng p2)
{
    double lat1 = (java.lang.Math.PI / 180) * p1.getLatitude();
    double lon1 = (java.lang.Math.PI / 180) * p1.getLongitude();
    double lat2 = (java.lang.Math.PI / 180) * p2.getLatitude();
    double lon2 = (java.lang.Math.PI / 180) * p2.getLongitude();
    double a = 6378.137 / 1.852;
    double dc = 1.852;
    double invf = 298.257223563;
    double EPS = 0.000000000005;
    double iter = 1;
    double MAXITER = 100;
    double f = 1 / invf;
    double r = 1 - f;
    double tu1 = r * java.lang.Math.tan(lat1);
    double tu2 = r * java.lang.Math.tan(lat2);
    double cu1 = 1.0 / java.lang.Math.sqrt(1.0 + tu1 * tu1);
    double su1 = cu1 * tu1;
    double cu2 = 1.0 / java.lang.Math.sqrt(1.0 + tu2 * tu2);
    double s1 = cu1 * cu2;
    double b1 = s1 * tu2;
    double f1 = b1 * tu1;
    double x = lon2 - lon1;
    double d = x + 1;
    double sx = 0, cx = 0, sy = 0, cy = 0, y = 0, sa = 0, cz = 0, e = 0;
    double c = 0, c2a = 0;
```

```

while((java.lang.Math.abs(d - x) > EPS) && (iter < MAXITER))
{
    iter = iter + 1;
    sx = java.lang.Math.sin(x);
    cx = java.lang.Math.cos(x);
    tu1 = cu2 * sx;
    tu2 = b1 - su1 * cu2 * cx;
    sy = java.lang.Math.sqrt(tu1 * tu1 + tu2 * tu2);
    cy = s1 * cx + f1;
    y = java.lang.Math.atan2(sy, cy);
    sa = s1 * sx / sy;
    c2a = 1 - sa * sa;
    cz = f1 + f1;
    if(c2a > 0.0) cz = cy - cz / c2a;
    e = cz * cz * 2.0 - 1.0;
    c = ((-3.0 * c2a + 4.0) * f + 4.0) * c2a * f / 16.0;
    d = x;
    x = ((e * cy * c + cz) * sy * c + y) * sa;
    x = (1.0 - c) * x * f + lon2 - lon1;
}

double mx = java.lang.Math.atan2(tu1, tu2);
double my = 2 * java.lang.Math.PI;
double faz = mx - my * java.lang.Math.floor(mx / my);
double crs12 = faz * (180 / java.lang.Math.PI);

x = java.lang.Math.sqrt((1 / (r * r) - 1) * c2a + 1);
x += 1;
x = (x - 2.0) / x;
c = 1.0 - x;
c = (x * x / 4.0 + 1.0) / c;
d = (0.375 * x * x - 1.0) * x;
x = e * cy;
double s = (((sy * sy * 4.0 - 3.0) * (1.0 - e - e) * cz * d / 6.0 - x)
            * d / 4.0 + cz) * sy * d + y) * c * a * r;
s *= dc;

double[] result = {crs12, s};

return result;
}

```

Now that we have calculated the current direction in degrees we can convert it to textual directions, like so:

```

if (b >= 337.5 || b <= 22.5)           output = "N";
else if (b > 22.5 && b < 67.5)        output = "NE";
else if (b >= 67.5 && b <= 112.5 )    output = "E";
else if (b > 112.5 && b < 157.5)      output = "SE";
else if (b >= 157.5 && b <= 202.5)    output = "S";
else if (b > 202.5 && b < 247.5)      output = "SW";
else if (b >= 247.5 && b <= 292.5)    output = "W";
else if (b > 292.5 && b < 337.5)      output = "NW";
else                                     output = "ERROR";

```

We wanted to make the directions more relative. Instead of “North, then West” we wanted to use “Turn Left, then Turn Right”. To do the I converted the textual directions from a number between 0 and 7, as below.

```
if(dir.compareToIgnoreCase("N") == 0)           output = 0;
else if(dir.compareToIgnoreCase("NE") == 0)      output = 1;
else if(dir.compareToIgnoreCase("E") == 0)       output = 2;
else if(dir.compareToIgnoreCase("SE") == 0)      output = 3;
else if(dir.compareToIgnoreCase("S") == 0)       output = 4;
else if(dir.compareToIgnoreCase("SW") == 0)      output = 5;
else if(dir.compareToIgnoreCase("W") == 0)       output = 6;
else if(dir.compareToIgnoreCase("NW") == 0)      output = 7;
else                                             output = -1;
```

The is subtracted the next direction number from the previous direction number to get the relative turn description, like so:

```
int prev = dirToNum(prevDirection);
int next = dirToNum(nextDirection);
int current = prev - next;

if(prev == -1 || next == -1)           return "EEK!";
if(current == 0)                       return "Continue on";
if(current == 4 || current == -4)      return "Turn Back";
if(current < 0)                       return "Turn Right";
else                                    return "Turn Left";
```

Ajax & RESTful Web Services

The use of asynchronous HTTP request (Ajax) was used to communicate between the client browser and the server to eliminate page refreshing and to provide an enhanced user experience similar to a desktop application. For communication between client browser and server, RESTful web services were used.

The following JavaScript code sets up the browser to use asynchronous HTTP calls. Various checks are made because some browsers implement this in a different way.

```
var routeHttp = getXmlHttpRequest();

function getXmlHttpRequest()
{
    var xmlhttp = null;

    try
    {
        // Firefox, Opera 8.0+, Safari
        xmlhttp = new XMLHttpRequest();
    }
    catch (e)
    {
        // Internet Explorer
        try
        {
            xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e)
        {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
    }

    return xmlhttp;
}
```

In Java you would create a Servlet to handle a HTTP GET request.

```
public class Routes extends http.HttpServlet implements javax.servlet.Servlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    {
        PrintWriter writer = response.getWriter();
        . . .
        writer.print(". . .\n");
        . . .
        writer.flush();
        writer.close();
    }
}
```

When making Ajax calls using HTTP GET then following JavaScript code is executed. A call back function is provided to retrieve resulting data because these calls are asynchronous.

```
getFromRestfulService(routeHttp, 'Routes', loadRoutes);

function loadRoutes()
{
    if(hasServiceResquestCompleted(routeHttp))
    {
        gRoutes = eval(routeHttp.responseText);
        createRoutesSelector();
        drawRoutesAndPOI();
    }
}

function hasServiceResquestCompleted(xmlhttp)
{
    if(xmlhttp.readyState == 4)
    {
        if(xmlhttp.status == 200)
            return true;
        else
            throw(xmlhttp.responseText);
    }

    return false;
}

function getFromRestfulService(xmlhttp, url, event)
{
    if(xmlhttp!=null)
    {
        xmlhttp.open("GET", url, true);
        xmlhttp.onreadystatechange = event;
        xmlhttp.send(null);
    }
    else
    {
        alert("Your browser does not support XMLHttpRequest.");
    }
}
```


Bike Route Editing

The main issue with this functionality is making the editing of a bike route intuitive and to have a well responding HMI. When editing routes a user can: add a new marker at the end or in the middle of bike route, delete any of the placed markers, and move any of the placed markers. Events needed to be trapped and a lot of visual/interactive testing was done to tweak how the HMI functioned and performed. There is an online video demonstrating this at: <http://server20.ielbcit.ca:81/mww-demo/>.

First we need to create a listening event. This even with handle: clicking on an existing marker (delete mark) and clicking on the map (add marker). The first marker is a special icon.

```
gEventListener = GEvent.addListener(gMap, "click",
function(marker, point)
{
    // check for marker type to avoid error message
    if(marker)
    {
        // does this marker exist in our list? if no exit
        var index = findMarker(marker);
        if(index == -1)
            return;

        // if the 1st marker is deleted then
        //setup the next marker as the 1st
        if(index == 0 && listRouteMarkers.length > 1)
            listRouteMarkers[1] =
                createEditMarker(listRoutePoints[1], true);

        // remove this marker from our list
        listRouteMarkers.splice(index, 1);
        listRoutePoints.splice(index, 1);

    }
    else
    {
        //place a new marker on the map and add it to our list
        listRouteMarkers.push(
            createEditMarker(point, !listRouteMarkers.length));
        listRoutePoints.push(point);
    }

    //render the route to the screen
    drawAddChanges();
});
```

The `createEditMarker()` function creates the markers and setup additional properties such as making the marker draggable.

```
function createEditMarker(point, isStarting)
{
    var marker;

    // make the fist marker "special" make sure the markers are draggable
    if(isStarting)
        marker = new GMarker(point, GMarkerOptions =
        {
            icon: redMarkerPin,
            draggable: true,
            title: "Click to delete or drag to move this marker."
        });
    else
        marker = new GMarker(point, GMarkerOptions =
        {
            draggable: true,
            title: "Click to delete or drag to move this marker."
        });

    GEvent.addListener(marker, "mouseover",
    function()
    {
        for(var i = 0; i < listRouteMarkers.length; i++)
        {
            if(listRouteMarkers[i] == marker)
            {
                gMouseOverMarker = i;
                return;
            }
        }
    })

    GEvent.addListener(marker, "mouseout",
    function()
    {
        gMouseOverMarker = -1;
    })

    // handle marker dragging
    GEvent.addListener(marker, "dragend",
    function()
    {
        listRouteMarkers[gMouseOverMarker] = this;
        listRoutePoints[gMouseOverMarker] = this.getPoint();
        drawAddChanges();
    });

    return marker;
}
```

We now want to render the changes to the screen so this function is called from the first snippet of code.

```
function drawAddChanges()
{
    gMap.clearOverlays();
    for(i = 0; i < listRouteMarkers.length; i++)
    {
        gMap.addOverlay(listRouteMarkers[i]);

        if(i)
        {
            var points = new Array();
            points.push(listRoutePoints[i-1]);
            points.push(listRoutePoints[i]);
            var line = createEditLine(points);

            gMap.addOverlay(line);
        }
    }
}
```

We need to setup the lines to handle click events so they can be split and new markers added.

```
function createEditLine(points)
{
    var line = new GPolyline(points, "#AA0000", 7, 0.50,
        {clickable: true});

    // handle adding markers in the middle of a route
    GEvent.addListener(line, "click",
    function(point)
    {
        for(var i = 0; i < listRoutePoints.length; i++)
        {
            //has the line been clicked on?
            if(listRoutePoints[i].x == this.getVertex(0).x &&
                listRoutePoints[i].y == this.getVertex(0).y)
            {
                //yes, so split the line at the point where clicked
                //and add a new marker
                var newMarker = createEditMarker(point, false);

                listRouteMarkers.splice(i + 1, 0, newMarker);
                listRoutePoints.splice(i + 1, 0, point);

                drawAddChanges();
                return;
            }
        }
    });

    return line;
}
```

J2ME & GPS Devices

There was a need to create a library/framework that would allow a mobile device to communicate with a Bluetooth enabled GPS device. With the absence of the J2ME Location API on the majority of mobile devices, querying the Bluetooth communication API manually had to be done.

```
//find all the bluetooth devices
int size = BluetoothManager.getInstance().find();
//loop through each device as find the holux gps device
for(int i = 0; i < size; i++)
{
    if(BluetoothManager.getInstance().getDeviceName(i).indexOf("HOLUX") > -1)
    {
        btDevice = i;
        break;
    }
}

//a holux gps device, read location
if(btDevice != -1)
{
    waitForm = new Form("Finding your location...");
    display.setCurrent(waitForm);

    BluetoothGPSInterface.getInstance().setDevice(
        BluetoothManager.getInstance().getDeviceName(btDevice),
        BluetoothManager.getInstance().getServiceURL(btDevice));
    BluetoothGPSInterface.getInstance().start();

    if(BluetoothGPSInterface.getInstance().getIsConnected())
    {
        GPGGAStrng location =
            BluetoothGPSInterface.getInstance().getLocation();
        boolean parsable = false;

        //wait for gps string data
        while(!parsable)
        {
            location = BluetoothGPSInterface.getInstance().getLocation();

            try
            {
                Double.parseDouble(location.getDMSLatitude());
                Double.parseDouble(location.getDMSLongitude());
                parsable = true;
            }
            catch (Exception e)
            {
                parsable = false;
            }
        }

        latText.setString(String.valueOf(location.getLatitude()));
        longText.setString(String.valueOf(location.getLongitude()));
    }
}
```

```

//send and sms to the mobiloe muse platform
//directions will be sent back via sms
String text = "clay FIND LATT " + latText.getString() +
    " LONGT " + longText.getString();
String address = "sms://+7785526873";

try
{
    MessageConnection conn =
        (MessageConnection)Connector.open(address);
    TextMessage msg =
        (TextMessage)conn.newMessage(MessageConnection.TEXT_MESSAGE);
    msg.setPayloadText(text);
    conn.send(msg);

    Form waitForm = new Form("You will get an SMS");
    waitForm.addCommand(exitCommand);
    display.setCurrent(waitForm);
}
catch (IOException e)
{
    e.printStackTrace();
}
}
}

```

Here is an example of the GPS data string that is sent by GPS devices and what the data means.

```

/*
Example: $GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

```

Where:

```

GGA           Global Positioning System Fix Data
123519       Fix taken at 12:35:19 UTC
4807.038,N   Latitude 48 deg 07.038' N
01131.000,E  Longitude 11 deg 31.000' E
1           Fix quality: 0 = invalid
                1 = GPS fix (SPS)
                2 = DGPS fix
                3 = PPS fix
                4 = Real Time Kinematic
                5 = Float RTK
                6 = estimated (dead reckoning) (2.3 feature)
                7 = Manual input mode
                8 = Simulation mode
08          Number of satellites being tracked
0.9         Horizontal dilution of position
545.4,M     Altitude, Meters, above mean sea level
46.9,M     Height of geoid (mean sea level) above WGS84
            ellipsoid
(empty field) time in seconds since last DGPS update
(empty field) DGPS station ID number
*47        the checksum data, always begins with *
*/

```