AJAX Project – Traffic Check

ICS 280 – Ubicomp for Post-Crisis Logistics

Phu Son Nguyen, Ryan Tang, Gary Suh
{nguyenps, cctang, gsuh}@uci.edu

21 Mar 2006

1. Initial Thoughts

The goal of this project was to incorporate two sources of data on the Web and a third source of data from the phone. The challenge was to find a meaningful and useful way for the different components to interact.

Originally, we had planned to use an existing source of information such as a registry of doctors or a list of restaurants and Google Maps. However, we found it was difficult to incorporate the mobile phone element in a meaningful way. The best idea we thought of was using Google Maps to find a desired location next to the user, or to use the mobile phone to allow users to give reviews. The problem with the first idea was that such functionality already existed. The problem with the second idea was that the AJAX map (web part) was not really necessary. Also we thought about getting away from location as one of the sources of data, but we could not decide what to substitute in its place. An intermediate idea was a kind of catchall map, insert a search and a location and it would give you the results in the area. Unfortunately this did not incorporate the mobile phone either and would shift too much focus to the AJAX web site.

Our final idea was to provide a user supported real-time traffic reports, inspired by Sigalert (sigalert.com), however leveraging on the Google Maps API with our own MySQL database. The main advantage of our system is it provides traffic for local streets. The mobile phone is used as a logging device for inputting local traffic information. Ideally, the mobile phone is GPS enabled, and all the user needs to input is the speed. Alternatively, the user should be able to enter the cross streets and speed and have the map determine the location. However, for this preliminary implementation, the user must input all the information manually. The map is also integrated with freeway traffic accident reports querying from Yahoo's database.

2. AJAX

The main component of this project is the AJAX web site. The AJAX code was based on Dan Theurer's tutorial on "How to build a Maps Mash-up" (http://www.theurer.cc/blog/2005/11/03/how-to-build-a-maps-mash-up/). The code was heavily modified and tweaked to integrate different data sources rather than just Yahoo!. Theurer's example includes the following features: Geocode an address and display on the map, overlay traffic information like accidents and construction zones, and overlay local search information. This is made possible by leveraging data available on the Yahoo! REST web service data sources. The original example overlaid all of this information on Yahoo! Maps utilizing Yahoo!'s own API. However after using Yahoo! Maps we found it to be a bit cumbersome at times. Therefore we decided to convert the whole application to overlay the same data sources on Google Maps using Google's map API instead. The conversion was not too difficult since we integrated both Yahoo!'s and Google's map APIs in parallel, plotting the same set of points from Yahoo! data sources on both.

The user's input data (see Phone Midlet section) is decoded into a URI string appended by a set of unique personalized parameters. This complete decoded URI and parameters are sent over HTTP to a server-side PHP script (write.php). The server-side PHP script, on the other end, acquires the appended parameters and encodes the given URI into a SQL INSERT statement that in turn is run on the back-end MySQL database. In other words, a row of data is inserted into the database's table.

Another PHP server-side script (read.php) handles the conversion from the SELECT query from the MySQL database and returns the query as a geoRSS formatted XML file. The web application JavaScript then parses the formatted XML file (from the read.php) and plots the given GPS coordinate sets on Google Maps. The JavaScript parsing of the XML file is made possible by two XML JavaScript parsing library: xmlw3cdom.js (http://xmljs.sourceforge.net/website/documentation-w3cdom.html) parser from SourceForge and xmlsax.js parser (http://xmljs.sourceforge.net/website/documentation-sax.html). The rest of the data associated with the GPS coordinates sets from the database is written in formatted XHTML to the sidebar of the web application.

The web application overlays data from a back-end MySQL database and also data from the available Yahoo!'s REST web services. Different color icons are used in order to distinguish different data sources the data is coming from. Three major cities have been added to the top toolbars of the web application for easy access. There is also an input box right next to the cities' names where the users can enter in any area specific string (street address and/or city, state) and view a map of that area.

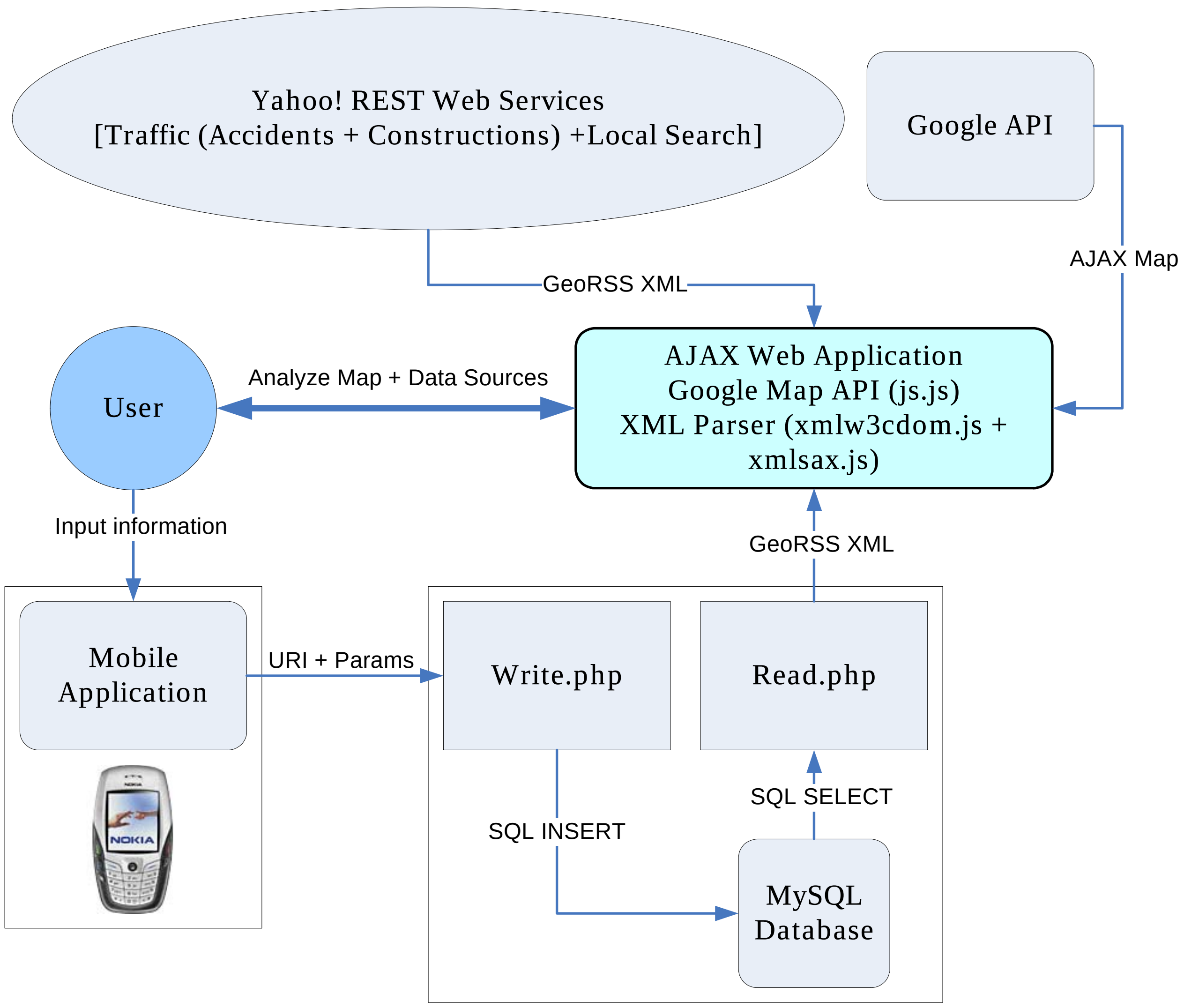

Figure 1: Overall System Architecture

3. Phone Midlet

The most difficult part of working with the phone was getting the development environment to work. Originally, we tried to use the Nokia Series 60 emulator however we were unable to build software for it. Next we tried using the J2ME KToolbar and were able to build and run applications in the emulator.

The midlet is relatively simple. It consists of TextField objects for: speed, latitude, longitude, street, cross street, and direction. Once the data has been entered, the user presses the "Enter" button. This causes the command listener to call a function to generate the proper URI string, connect to the database, and send the information. We used an HTTPConnection for this project. Originally we planned to establish a connection then transmit the data using a DataOutputStream. However instead we decided to write a PHP file on the server-side that generates the request to the database when given the proper parameters as part of the URI.

Writing the code was relatively straightforward. The only problem we ran into was when we first tried to create the connection. The phone always seemed to hang at the connection verification screen. Eventually we found the solution was to create a separate thread and run the connection in that thread. The problem was when the connection was established it would take over control from the command listener and when the connection was closed the command listener would not continue running.

4. Conclusion and Future Work

The web aspect of the completed project looks fairly well polished, offering traffic centered at three different locations, Irvine, Orange, and Los Angeles.

Some potential improvements and uses we discussed include:

- Discard input after a certain time threshold.
- Incorporating Sigalert (Sigalert.com) and/or other freeway data.
- Automatically acquire latitude and longitude from GPS on the mobile phone device with onboard GPS receiver.
- Calculate locations from cross streets, not just coordinates.
- Calculate routes based on real-time information rather than static estimated fastest route.
- Draw poly lines for each user’s session of inputs.