BusTracker design document

Pekka Ihalainen

1. Abstract

BusTracker is a interactive web-based software for monitoring busses used in public transportation in Tampere, Finland. The aim of the project was to give insight of where and how busses move in Tampere using GPS coordinates displayed in Google maps and OpenStreetMap. Software was designed to be run on any modern browser including but not restricted to Opera, Internet Explorer, Safari, Chrome and Firefox.

The project is part of university course TIETS16 Programming Project and it is held by Professor Jyrki Nummenmaa. The project was supervised by Paula Syrjärinne and Tero Piirainen.

2. BusTracker

BusTracker is a software, which displays the current locations of the all the busses of Tampere region in a selected map type. Map types currently supported by the software are Google Maps and OpenStreetMap. BusTracker reads JSON output, which holds all the data of the busses in a single "file", from the SIRI access interface provided by ITS Factory. Using that JSON data, the locations of the busses parsed and displayed to selected map type. Refresh rate of the rendering is limited to maximum speed of one second. This maximum limit is set by ITS Factory's SIRI interface, which only refreshes its data every second but also the browsers (particularly mobile-browsers) can't handle faster rendering rate, since the parsing and processing of the data are quite CPU and network heavy jobs.

Currently BusTracker supports number of different features. Below is the list of the features with the explanation.

- Refresh rate: User can change the refresh rate of the bus tracking. Available options are one second, three seconds and five seconds. The default setting is five seconds and it is recommended for slow computers or mobile browsers. The fastest, one second option, is only recommended with the modern computers with broadband network.
- Bus line: User can choose which line is to be tracked on map. The list is taken from the TKL's website. Options from which to select are one of the TKL's provided lines or all lines. Currently software doesn't support displaying of two different lines simultaneously.
- Real-time tracking: User can toggle the tracking of the busses on and off. When the tracking is off all the data previously gathered are saved however if user resumes the tracking and user had tracing on before pausing the tracking, the tracing continues from next possible gps-point and thus making the tracing line jump weirdly.
- Tracing: User can toggle on and off the tracing of the busses. If the tracing is on, all the data coming to system is saved and that data is processed and shown as a tracing line for particular line or all the busses (depending on bus line option). Note that if tracing is on for a long time, the data stored by the software increases dramatically thus making mobile browsers laggy and in the worst case quit unexpectedly.
- Map type: User can select from two different map layouts which one to use. Currently supported maps are Google Maps and OpenStreetMap. When switching between the maps, all the settings and viewport coordinates are saved and brought to new map type. Changing between maps should be transparent.
- URL parameters: User can change the default settings using URL parameters. Currently supported parameters are zoom, latitude, longitude and map. With these options the software can be set to

display the desired location with the zoom level and map type. Convenient when pasting the URL to another person.

There we also several suggestions for feature which couldn't be implemented at this point. Below is a list of some of those and reasoning for not implementing.

- HERE Maps: 3th map type for the software to display. Reason this didn't get implemented was that coding language (GWT) didn't have support for HERE Maps. Google nor any 3th party didn't provide bindings for HERE Maps javascript -> GWT.
- KML layer for bus lines: A full KML layer for every bus line was implemented but removed because it didn't provide any useful information at that point. KML layer for single line for suggested but it was left outside of this version due lacking of time. Parsing GTFS file for single bus line is time consuming if it is done 'on-the-fly' so intelligence server-side implementation would have been needed to accomplish that.
- Bus stops layer: To render bus stops on the map. Basically the same explanation than in KML layer case. However it could have been achieved with less for if the bus stops would have been hard coded into code.

3. Technical details

3.1. Tools and requirements

BusTracker has been coded using GWT, Google WebToolKit, which can be found at http://www.gwtproject.org/ . BusTracker's code is plain Java code with CSS and HTML files. GWT compiles the java code into javascript code and packages all the files into neat pile. From the end user point of view all the software is, is a HTML file and several javascript files with one CSS file. Because of using Java with GWT, the software in theory can be ported to native java also.

BusTracker runs on any web-server which has PHP5 support. PHP support is needed because BusTracker uses PHP as server-side language to retrieve the JSON data from the ITS Factory SIRI interface. Apart from this, BusTracker runs solely on client-side.

BusTracker utilizes Google Maps and OpenStreetMap public interfaces to display the map data. Those interfaces are for Javascript so 3th party bindings has been used to integrate these services to GWT's Java code. GWT OpenLayers API (www.openlayers.org) is used to display OpenStreetMap and gwt-google-apis (http://code.google.com/p/gwt-google-apis/) is used to display Google Maps. Note that Google Maps could be displayed with OpenLayers also however OpenLayers doesn't support all the features at the same scale as the gwt-google-api.

3.2. Architecture

BusTracker uses traditional MVC, Model-View-Controller, architecture in its design. BusTrackers model handles the JSON data retrieval and processing as well as keeps the settings such as zoom level, latitude, longitude and viewport position.

Views are created and destroyed whenever demanded. If the program starts with Google Maps, the GMView is created and loaded. When user switches the few to OSMView the GMView gets destroyed and OSMView is created. This way the views release the memory they consume when different kind of bus data is processed. Basically all plotting and tracing data is kept inside view instead of model. Reason for this is the difference between Openlayers and google-maps APIs. It was more efficient to make each store the data in their own data storages rather than try to unify them.

Even though controllers are usually own objects or classes, they are in BusTracker implemented inside view. GWT uses event handlers for handling events from elements and mouse and therefore it is more convenient to make handlers directly into view rather than put them all in different class.

Model uses pull method when communicating with the views. Model sends a event-message to every and all observers and the observers, in this case the views, decide if they want to act upon that message. If they decide to act, they make a method call to model to retrieve data they want.

Model retrieves the JSON via server-side PHP file at the speed the user has requested. The retrieval itself is not time consuming process since its non-blocking request via GET. However parsing the JSON and processing the data takes time and CPU power. This leads to biggest problem the software has. On average the processing took from 500ms to almost 800ms. This leaving a cap of 500-200ms for model to give other methods and software time to work. It was soon noticed that with 500ms of free time for browser to handle other requested than models requests, the Google Maps and OpenStreetMaps rendering was laggy and controlling the maps was hard because from time to time the events didn’t get passed to maps event handlers. With three of five second update speed this wasn’t a major concern. With those speeds the maps had plenty of time to handle rendering and scrolling but anything faster than that made the software laggy. One solution to solve this problem would be to use threads. However threads are not supported in natively Javascript at the moment. And any kind of other threading required native Javascript API which isn’t supported by GWT. At this point the problem remains within the software. If used in slightly older computers the rendering of the maps is laggy, in modern computers it is competent but not ideal.

4. Conclusion

Overall the project in my mind was a success and achieved what was required of it. At the very beginning I didn’t know much about GPS nor anything to tools and technologies related to that. During the project I feel like I grew at least small understanding of how those things work. It was one of the most educating projects I have done during my studies and I liked it a lot for that.

If the project is at some point taken over by someone else, in my opinion the main areas of improvement would be, solving the problem with update speed, making the markers look good and adding bus stops feature.

Of course if at some point the GTFS from TKL could be imported into Google Maps itself it would make great addition to software. Then it would be possible to see bus lines properly and watch where the busses are along those lines.