CSCE 487 Final Progress Report
Team 4 - Back-end Development

Objectives

Our high-level objective for this project was to support the development of mobile applications for the StarTran public transit system. The planned applications will allow users to better utilize the transit system by integrating bus locations and static route data to generate optimal or near-optimal routes for travel by foot and by bus. However, the limitations of mobile devices (battery, computing resources, network, etc.) and the complexity of designing these routes suggests the need for a server.

This semester, we have created a web service that has abstracted away these problems for the mobile teams. This service has the added benefit of reducing code duplication; since each mobile platform has such a distinct environment, it would ordinarily necessitate three differing implementations of the route-finding routines. Our service provides a unified routine with one design, one resource set, and one set of bugs.

What

Our server, Daphne, has been progressively implementing new services throughout the weeks since her initial release. One of the biggest services she has provided is a directions service -- which provides data about a bus route to take given a starting and stopping location. Right now, the algorithm selects a route based on the shortest distance from the current location to a bus stop. Because the route has more than one possible “traversal” from one bus stop to another (i.e. ways the bus can get from one bus stop to another), the algorithm then searches for the optimum traversal within that route, and that traversal is returned to the phone teams. One major improvement that can be made to this algorithm is in the selection of the bus route. Obviously, a major flaw in our current implementation is the lack of accountability for the walking distance from the second bus stop to the user’s destination. An improvement to our algorithm would involve calculating the distance from the user’s current location to the first bus stop, calculating the distance from the second bus stop to the user’s destination, and then adding these two values together to find the total walking distance. We would then use the bus route that had the shortest walking distance. A further description of our development process of this algorithm is described in the ‘How’ section below.

Another major functionality that we are almost finished implementing is the ETA functionality. This provides an estimate of when the bus will arrive at a particular location based on the route history. Route history was found by monitoring the buses on all routes for an entire iteration of each bus over the route. The algorithm for gathering this data included obtaining the position of each bus every 10 seconds. As the position of each bus was obtained, a programmatic graph of the bus’ path was constructed. This graph was stored in our database and retrieved when we wanted to find information about buses for a particular route. The graph provided an easy way to traverse through the various bus locations along a bus route and retrieve bus position at a particular time.

How

In terms of our optimum bus route algorithm, the biggest reason why we did not optimize our algorithm in the manner described in the 'What' section above was because our focus was on fixing other difficulties we ran into with Daphne, such as getting data for the bus route algorithm to store in the database properly, as well as figuring out which waypoints to display to the phone teams for the bus route. The problem that we had with storing data in the database was of particular importance because this data was used in calculating the optimum route traversal that we returned to the phone teams. If an optimum route traversal could not be returned, we were unable to return data to the phone teams.

Because this issue was so important to fix in a timely manner, we invested much of our time and energy into it. We tried several different things, including making alterations and modifications to our algorithm, playing with different techniques of storing data in the database (such as creating and loading YAML files), changing our server settings, and creating SQL dump files to try and load the data into the server manually. One particularly confusing issue regarding this problem was the fact that, on our local machines, data was stored in our local databases just fine. However, for some unknown reason, the instance of our code that existed on the server did not store data properly.

We never figured out the cause of this problem, but our solution was to inject data from one of our local machines into the server's database by using SQL statements. This was similar to the SQL dump files we tried loading into the server's database. However, we had to modify the commands in these dump files in order to make them load properly into the server's database. In summary, finding a solution to this problem was time-consuming and fairly frustrating. It set us back a little in terms of providing some of our data to the phone teams. This made it so that we had less time to receive feedback from the phone teams about the data we were sending them.

By the time we fixed the aforementioned problem with our database, the phone teams had about one and one-half weeks to provide us feedback about the data we returned to them. As it turns out, only one of the teams (the iPhone team) was far enough advanced in this particular area of functionality to provide us with feedback in a timely manner. The feedback we received from them involved a request for the re-ordering of the route waypoints we were returning. The reason they made this request was due to the particular way they were rendering the data we gave them to their map.

Because of the way they chose to render this data, they needed the waypoints to be in a particular order. We spent some time working with the iPhone team in attempting to alter our data output format so that it was compatible with they way they were rendering it. It turned out that this was somewhat of a time-consuming process, as altering our data output involved four steps: making the modification to our output algorithm, deploying this new modification to the server, having the iPhone team test our output on their device, and receiving feedback from them.

With these two pressing issues at hand, we did not have a chance to optimize our bus route algorithm to search for a more optimal bus route. However, we believe this will be a fairly straightforward change, and we plan on implementing it before handing in our code base.

Quality

Our testing and evaluation efforts remain less comprehensive than we'd like. A suite of tests to assess the functionality of our system is in place, and is run before each release to ensure that the new version's changes have not disturbed old features. However, there are less tests than we would like, which could present an irritation and hindrance to the future developers of Daphne. Our development process and the nature of our project do not lend themselves to comprehensive testing, but more could be done to assure developers of the robustness of old systems and new changes.

One component of our system that exemplifies successful testing is the static schedule feature. We have a battery of functional tests to exercise the static schedule, which are easily available by accessing the /test directory when the app is running in testing mode. The results of having these tests being in place are just what we hoped -- very boring! No problems have arisen, but if they do, we'll know quickly and without a doubt.

Risk Analysis and Mitigation

Since our application is mainly used as a service for the mobile phone teams, we are largely reliant on them to find bugs and other inconsistencies in our code. To counteract this, we still have JIRA set up and accessible by everyone in our senior design class for which they can log on and submit a bug report. We also have email notification set up so that as soon as a report is generated we can assign the issue to a respective teammate. So far we have had 13 issues submitted but unfortunately not many of them are in regards to how the directions function is actually performing. The mobile phone teams are always encouraged to submit a JIRA report, day or night.

With this project continuing to be maintained and further developed upon even after this class finishes, how to transfer between hands was needed to be taken into consideration. Using fully commented code will help with the understandability for the future owners to continue development. Also since the project is developed with the MVC pattern in place it will ease transition for anyone that has used this structure in the past. In addition, JIRA but reporting will continue to exist into the future for fleshing out bugs found during testing. The distributed nature of Git will allow future developers to easily import our entire source repository, including all history and past branches.

We have also been largely reliant on a few eternal services that have caused problems for our development practices. More than a few times, the TransitServer we have been querying has gone done due to a number of events mainly from StarTran and the North Carolina company

they have their service supplied through. This has caused testing for both our team and the mobile phone teams to become halted numerous times. In order to mitigate this risk, Paul has met with StarTran and his server will be able to access the raw AVL data coming directly from the North Carolina company instead of going through StarTran as a middleman. This will cause the service he provides to be much more dependable and he will also have access to much more information. There has also been discussion of hosting Daphne on a more stable server than csce.unl.edu before the BusLinc applications go live.

Another external service that has a large part in most geolocation applications is Google. We have been querying data from them for directions services and geocoding locations for the conversion of addresses. Due to this we have been exceeding our quota limits of 2,500 requests per day from the same IP address. This is still an occurring risk but action has been underway to retrieve a non profit grant from Google for their Google Earth Outreach program. This would allow our server to query Google without a quota limit, which would greatly help when we are making many requests per day once the applications are available to the public. An alternate option that could be implemented is client-side geocoding, with the individual's phone making a request to Google instead of our server which would help to alleviate much of the queries we are making to them.

Conclusion

We are pleased with the web service we have created for use by the mobile teams, while cognizant of the fact that more functionality could and will be implemented. Our service has provided a valuable starting point for the mobile teams to develop upon. Most functionality was implemented before major delays were caused for the mobile teams; this was a major concern for us in the planning stages, as we were put on the "critical path" with very little knowledge of the system we were going to implement or how we would begin to build it!

We feel the agile methodology adopted in the early part of the spring semester was a major contributor to our ability to keep up with the evolving needs of the phone team; as the 'Agile Manifesto' reads, "Responding to change over following a plan." This process, along with Mr. Bennett's guidance, enabled us to take on very small tasks and begin acquiring the experience and assembling the functionality necessary for this project. Every week, each team member would be assigned an objective to complete; the next week, the accomplished objectives would be reviewed, the remaining objectives discussed, and new objectives assigned. Frequent contact with the phone teams kept us abreast of problems in our code and what new issues were most critical to tackle

Some specialization naturally occurred in this process; for example, Trevor became the go-to guy for problems relating to our repository and for communicating with the Yahoo and Google APIs, Peter managed the route representation and the process of finding the shortest path from one point to another on a bus route, and so on. In some ways this specialization certainly sped along the development process; in some others, however, it led to misunderstandings and assumptions during development about how other parts of the system work. All in all, this

specialization was likely a net gain, but certainly a trade-off.

The preceding two paragraphs begin to describe what our team has learned in the course of this project. Another aspect we learned about was requirements finding: what kind of product do our users want? Knowing this, what kind of features will the phone clients need? These two questions were surprisingly complex and the answers evolved throughout the course of the project. We had to learn how to share code harmoniously (in our case using Git). We had to learn how to deploy our code, and that deploying can entail a lot more than it seems at first glance -- configuring databases, making backups, managing the inevitable case that the hosting server suffers downtime.

All this is, of course, in addition to the new experience with the technologies at play here -- JSON, Java, the Play framework, SQL, and so forth. Many of the technologies we learned here are almost sure to be applicable in future development as web services grow in popularity.

Due to the succinctness of the Play framework and the structure of our codebase, we anticipate that Daphne will easily be able to change hands to a new development team. The next team may take Daphne in any direction they wish: certainly, a likely start would be a more extensive pathing algorithm, support for transfers, and keeping up on the demand for new features from the developers of the phone platforms. Supposing new transportation data were available, there's no reason Daphne couldn't also be ported for use in other areas.