Today we’re going to present to you a generic overview of the entire project in terms of the flow of code , how we handle data and in what format. This will also provide insight to the semantics group detailing what file format we read from and write to. Also, I would like to point out the the names of the services as we call them slightly differ from the names that the semantics group uses. For example what we call AllPathsFinderService, the semantics group calls it route\_service but once we get to integrating everything we’ll take care of it.

Before we get to explaining anything, I’d like to define a a few terms.

path vertex - A path vertex is a vertex in the graph representation of the map.

For example A, B, C …up to U are the path vertices

Link - A link is a straight line joining any two adjacent vertices.

Waypoints - Points between each pair of adjacent path vertices, interspaced by a

distance of 5 meters. The 5m resolution is a choice based on our intuition.

All the Red Pins that you can see are all waypoints between J & K interspaced by 5 meters

Okay , so the first slide is an updated version of our map which is in terms of a graph. The user is no longer constrained to start at a specific known location and end at a specific location. This was a constraint that we spoke of , last time. But with the development we’ve made over the week the user can get the most efficient path from any vertex on this map to any other vertex. An important note to make here is that as you can see, the link joining any two vertices is a straight line. This is of importance to us as I’ll explain when we talk about a service called HistoryService.

Okay.. so here’s the general flow of code as we have discussed until now.

For example, the user may wish to go from vertex A to vertex G. The user now wants to know the most efficient path from A to G. To do this to User application divides this task into 2 sub-tasks.

1. The first sub-task is to Find all paths between start and end point in terms of coordinates.
2. And the second sub-task is for each path, it needs to request for the ambient light values along each link forming the path.

For the first sub-task the user Application is going to perform a service discovery and look for a service that can find all possible paths between the start and end point.

The user application should be able to find a service called AllPathsFinderService in the cloud which does this job.

We’ll take a look at the source code for this service.

----------------------------------------------------------------------------------------------------

This service accepts start and end coordinates as inputs from the user Application.

This cloud service makes use of two static resource files that it stores locally. First, is the adjacency list of the graph.

Second, the coordinate values of each vertex.

For the first part of the code, it’s going to read the adjacency list from a csv file, adjacency\_list.csv.

The adjacency list represents the entire graph. Now that the code knows what the graph looks like because it read the adjacency, it’s going to interpret the start and end coordinates in terms of vertices to generate all possible paths between the start vertex and end vertex. Now you see this runs purely a graph algorithm with no notion of coordinates. And thus, the output of this segment of code is going to be lists of vertices that make up each path. Let’s run this code and see what I’m saying.

[Play video of all\_paths.py generating all the paths]

Now we have all the paths in terms of vertices but this is not what the service is going to send back to the User. The user Application had sent start and end points in terms of coordinates and it so it also expects a reply in terms of coordinates. The user Application has no notion of what the vertices mean, it only understands coordinates.

Now, the second segment of this code is going to translate these lists of vertices to coordinate values. How can it do this ? Using the second resource file that we spoke of which specifies the coordinates of each vertex.

Let’s run the code and see what we get.

[Play video for 2nd part of code.]

Here’s the output as a csv file. This all\_paths.csv file will be sent back to the userApplication as a reply to the userApplication’s request.

Looking at this interaction from the perspective of the user Application; The user Application specified start and end coordinates and it received a reply from the AllPathsFinderService , a csv file which has all the paths in terms of coordinates.

Our first sub-task is now complete.

Moving on to the next sub task, we need to now request for the ambient light values along our paths of interest. For this purpose the user Application again needs to do a service discovery and look for a service that can provide the ambient light values in our area of interest.

The user application should be able to find a service called HistoryService in the cloud which can do this job.

Simple as it sounds, the userApplication will forward all the paths in terms of coordinates to the History service to get the ambient light values along each path.

Before we move ahead, let’s talk a little bit about the History Service. This service runs in conjunction to the Polling service. The HistoryService maintains a grid of predefined resolution let’s say of 5 meters. And so any value it receives from a userApplication is quantized to the nearest point in this grid and the ambient light value at this point is updated. So in short the History service maintains ambient light values at each 5m interspaced points of the grid. Also, recall that all the links joining any two vertices is a straight line.

Now when the user Application requests for ambient light data, it is sending a list of coordinates. The history service is going to look at this data in terms of links. For example, the first and second coordinate in the list forms a link. The second and third coordinate forms the second link and so on.

So when the History service interprets this as links, it’s going to go look for average ambient light values along each link. Now when I say average. This is an average over the length of the link and not average over time. A fair question to ask would be, does the history service only store average ambient light along each link ? That’s a very poor history service. So the answer is no. The history service maintains history at a much higher resolution. It maintains the ambient light values for all the points on the map that are interspaced by 5 meters and these grid points in short are your waypoints.

Note that this resolution of 5m is invisible to the user Application. If the HistoryService maintains data at a resolution of 1m or 2m.. It’s a better service. If it has poor resolution it’s a bad service. In either ways, the UserApplication does not need to know this resolution.

The HistoryService will sum up all the 5m interspaced values along a link and divide by the length of the link to provide an average ambient light value of the link. In such manner the HistoryService will compute the average light values for each of the links on a particular path pack them in a list of numbers which represent average ambient light values. This will be repeated for each path.

The User Application will now receive this list of average ambient light values on each link along a path and may choose to deal with this data in anyway it likes. You could be running an application which only monitors this data. Maybe you’re studying the how population density relates to road lighting conditions. For our application we are finding the most energy efficient path and so we must find the path on which least energy is spent. In short this is the path which has the highest average ambient light over it’s length.

The user application will thus do this averaging to find out the best path.

Note the distinction between the averaging that takes place inside the History service and the averaging that’s done at the user Application.

The averaging done at the HistoryService averages out the values over the resolution of waypoints because the waypoints is an abstraction inside the service. The output required is only average ambient light over a link.

The averaging done in the User Application however is for a specific purpose of most efficient path calculation

Once the userApplication has found the most efficient path it will request for navigation along the path.

The navigation is a separate module altogether and so we won’t discuss that here.

In summary, the user Application sent start and end coordinates to AllPathsFinderService.

It received a reply of the available path options in terms of coordinates. This data was sent to the HistoryService to retrieve information on the average ambient light along each link forming a path.

And the UserApplication finds out based on this information , what the best path is.

Some points that I would like to add which I did not address during this explanation is does our system use live data too ?? Or does it simply depend on Historical data ? For our first implementation of this project we are dealing with only historical data. The usage of live data will be an addendum and if sufficient testing deems it possible to use it, we will try to incorporate that in our project.

So that’s pretty much everything that happens in the cloud.

Thank you.

* Service Discovery required to find HistoryService.
* All Paths in terms of coordinates will be given to HistoryService
* HistoryService maintains grid of 5m resolution
* HistoryService interprets input list of coordinates in terms of links