Temporal Route Interactive Planning System (TRIPS)

Assignment 3: Low Level Design & Work+Test Plan

Introduction to Software Engineering
Department of Computer Science
Technion – Israel Institute of Technology

Winter 2010-2011

1. Goals
In this phase we will simplify and define a subset of all the requirements. This will be
called build 1, the details of which will be discussed later on. You will produce a low
level design and a test plan for this build. You will be required to prepare a document
containing these two sections:
1. Low Level Design: will include the Low Level Class Model and the Sequence
   Model.
2. Test Work Plan: will include a description of your plans to see the successful
   execution of the test for build 1
3. Test Design: will include a description of the unit, integration, and acceptance
   you intend to run along with their expected results.

2. Responsibility Distribution
The tasks should be distributed uniformly among the members of the group. You are
required to describe the responsibility of each member in executing the phase. Define
the role and actual work of each member, in the level of functional responsibility:
each member is responsible for some modules and/or functionality and perhaps some
development task, such as document integration, internal reviews, etc of the system.

3. Submission
In addition to the Low Level Design, the Test and Work Plan described in the project
goals you should also provide the previous submissions which include the updated
requirements document and the high level design document from previous
submissions. These will not be graded but will be used to check consistency. If you
need to make any changes in these documents please make sure you also provide a
change log document describing your changes. The document (including previous
phases) should be printed and submitted as one coherent document containing
multiple sections, and also submitted electronically in pdf format via the course
website. The submission should also include a cover page including the group name,
the names and ids of all group members, and also the number of cell to return the
checked exercise to. You should use the RoboLib documents as a reference for the format of the submission (diagrams, tables etc.).

Please inform Roy (royl@cs) of any problem, and in particular Miluim, as soon as such issues arise.

3.1 Low Level Design

The section should include the following components:

- One or more Class Diagrams. It should be clear from your design how your system is organized. When designing your classes you should also think of design solutions that will allow you to progress to build 2 (the next version of the software that will include more of the original requirements) with minimal changes to your code. Consider using relevant design patterns for this purpose.

- Each Class Diagram should include a class description table that contains:
  - The class name
  - A description of the class (including its role and responsibilities)
  - Class members (attributes and methods) including a detailed description. The methods should also include return values and parameters.

- Sequence diagrams that cover all the use cases that are relevant to build 1.

3.2 Test Design

The test design should include the following:

1. The Acceptance Test - this is basically the black box system tests that you intend to perform on the system, to show its correctness. The tests should be thorough and convincing. It should indicate that your system is coherent with the requirements. The document should be formatted in the same manner as the test procedure document from the black box testing tutorial. The table should contain the following 4 columns:
   - Requirement numbers
   - Verify that
   - Test description
   - Expected result

2. Unit tests – for this part you should pick at least 5 of your major classes from your detailed, low-level, class diagram (classes that have meaningful, non-trivial methods) and specify a test plan for them. You should describe the following:
   - Stub (mock) objects you intend to implement so that you can test these classes in isolation.
   - Sequence of method calls and description of the class’s expected state (the described state should be testable – keep in mind that in the next assignment you will write the actual unit tests that will realize this description).
3.3 Test Work Plan

The work plan should include the following:

1. Define how you plan to perform your integration tests. A possible solution is to perform the integration tests by creating the system before all its components are complete and to begin running the acceptance tests that you expect to be successful. In addition, you may write addition unit tests for the purpose of testing a combination of a group of classes, that is, not each class in isolation as with the standard unit tests.

2. Define when you plan to begin the integration tests (you should specify a deadline), Keep in mind that a good plan will not delay the integration phase until a few days before delivery.

3. Define who will be in charge of running the acceptance tests during integration and as the final integration phase. You do not need to specify an individual’s name, just say if it will be one person in charge of QA or each team member should run the acceptance tests relevant to his code. You may decide on different options as well.

- For this section you should note that the implementation phase of build 1 must be submitted by the 23rd of January 2011

Build 1 Design

In Build 1 we focus only on the application that will be running on the server side. Hence, for the purpose of developing this build, you will need to design a proper interface for the server and also design the server’s code. This interface will allow the server to serve a client that will be implemented in future builds. The interface needs to be full in the sense that it must be able to provide to a client application any information the client may need according to the requirements which are detailed in the client story. For this purpose you must be aware of any calls a client can potentially make to the server, what type of information it passes and what type of information it expects to receive from the server. **Note that any requirement that does not affect the server can be ignored for the purpose of this build.** When implementing this build, i.e. in assignment 4, the interface you designed will be XML based. That is, it will receive all its input via XML documents and return all its output via XML documents. In assignment 4 you will define the structure of your XML documents. These will correspond with the interface you define in assignment 3.
TRIPS is an advanced route planning system which enables mobile users to enter their route requirements in the form of a query and receive route instructions in an interactive fashion. The system is comprised of a client application running on Android phones and a server application running on a cluster of backend servers.

Recall that all requirements related to the client are not to be implemented in Build 1. The interface you design for the server should allow future implementation.

The client application is the front-end of the system and it provides the interface between the human user and the actual search. Upon loading the application, a map of the area centered around the device’s current GPS reading is retrieved from the server. There are 5 menu items allowing the user to (1) set a destination, (2) add stops, (3) set constraints, (4) embark on the route or (5) quit. Initially options 2, 3 and 4 are grayed out until the user selects a destination. The user can quit the application by selecting item number 5.

The user can then begin to create a route search query by selecting to add a destination. To specify a destination, the user taps on the “set destination” item which, in turn, opens a text box that allows the user to specify a search term, for example “Motel 6”. For the purpose of Build 1, you may assume that this search term is just a name of a category from a list of possible categories that will be detailed ahead. A map zoomed and centered around the 10 nearest locations to the device’s GPS that match the user’s search term (in this build, have the same category as mentioned in the search term) will be retrieved from the server. These potential destination points will be marked with a flag sign. The user can obtain more information about a potential destination point by slightly tapping on it. The user can select an actual destination by tapping on it for a longer period of time. Once a specific destination is chosen, the other flag signs are removed and the selected destination is highlighted. Note that the route’s source location is defined to be the GPS reading at the moment a destination is selected. After setting the source and the destination of the route the user may now choose to add stops by tapping on the corresponding menu item. Tapping on the menu item will open a text box allowing the user to specify his search
Once more in Build 1, you may assume this search term is a category from a list of predefined categories. After doing so, the server will retrieve no more than 25-5 locations of entities that match the search term and that are within an ellipse with the source and destination locations as its focal points. The radiiuses of the ellipse are calculated using a formula that will be implemented by a third party. For the purpose of Build 1, given a start location $s$ and a target location $t$, the ellipse will contain all objects $x$ such that $d(s, x)+d(x, t)<1.25*d(s,t)$ where $d$ is an Euclidian distance function between locations. Note that there is no requirement that this set of points be calculated efficiently. If more than 5 objects exist in this ellipse you are to select those with the smallest $d(s, x)+d(x, t)$ value. A map that is zoomed and centered according to the ellipse will be retrieved from the server. For each stop specified by the user, the relevant locations will be marked on the map using a unique color for each of the stops. The user’s query, as it is being created, will appear in the top left of the screen. In addition to the add stop menu item the user can also choose to add constraints by selecting the set constraints item. As a result, a dialog will open allowing the user to specify a time interval by which any of the stops or the destination should be visited. The client application must be able to provide a response to the user instantly. For any of the above operations that require communication with the server, response time should still be within 250 milliseconds (in Build 1 network communication time can be neglected).

When the user taps on the embark on route item, a query containing all the relevant information, is generated in xml format and sent to the server. The server processes the query and uses a third party algorithm to calculate the directions leading from the user’s current position to the first waypoint in the route. In Build 1 you need to implement a stub algorithm (any reasonable simplistic approach will do) that merely attempts to find a feasible solution. There is no requirement that the route calculated by the algorithm is optimal in any way but rather only that it strives to find a route that adhere to the constraints specified by the user. If a route is found, the client directs the user to his next location in a similar manner to standard vehicle navigation devices. That is, the path is highlighted on a local map and driving directions are given vocally. This will be a separate software component that will be provided by a third party vendor. When the user reaches his next location, the client will display a small textbox asking if the user is satisfied with the visited entity. The
user’s response will be sent along with the query back to the server for calculating the next location to visit. When the user reaches his final destination it is guaranteed that he visited exactly one entity that satisfied him with respect to each of the search terms defined in the stops. It is also guaranteed that each entity has been visited within the time frame specified in the constraints. Note that data sent from the client to the server or back must be encrypted. For Build 1, when the server needs to return a map to the client, it is sufficient to return the relevant list of nodes and edges (roads) between them along with any relevant information. There is no need to generate an image of a map. For the purpose of this build we assume the client can do so using the information it receives from the server.

The geographical data for Build 1 will be taken from the following URL: http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm. In this website the title “California's Points of Interest With Original Category Name” contains all the points of interest required for Build 1. Your application does not need to read this data from the internet. You may save it to a local file in advance. Note that this data does not include road information. For Build 1 you are to create a fake road network yourself by simply assuming that a road exists between every pair of nodes. The road’s length is simply the Euclidian distance between the locations of the two nodes it connects.

The server maintains statistics about each entity indicating the ratio of user satisfaction for a given search term. These ratios are then used by the third party algorithm for finding a route with the minimal expected travel distance.

The client also supports an automated software update option. The client periodically checks with the server to see if there are software updates ready to be installed. If there are, the client informs the user, and if the user concurs, download and installation begins (in Build 1 there is no need to send a real updated version of the client, sending some predefined data representing a possible client update is sufficient). The development team will use a third party tool that allows them to publish software updates, by version, that will be shipped to the clients using the mechanism described above. For maintaining system code, the development team should use SVN as their version control.