1 Overview

The GPS and wireless technologies have reached a point where it is possible to monitor the locations of people, vehicles etc at a very fine granularity fairly cheaply (e.g. some cellphones already have inbuilt GPS devices that are accessible to the user; look at http://www.accutracking.com/). This is already used quite heavily for vehicular applications such as cabs, shuttle services (UMD shuttles are scheduled to be outfitted with GPS within next year) etc. We will explore one such possible application in this project.

The main goal of this project is to design an end-to-end application that allows storing and interacting with the time-varying location information about registered users of the system. The location information is assumed to be collected using some GPS device, and should be stored in the latitude-longitude format. The database should also store information about interesting “locations” (e.g. cafes). Finally a simple social network (based on a symmetric friend relationship) should be maintained. More details on the kind of data to be stored, and queries to be supported are provided below.

Though we will not focus on them, note that there are many privacy concerns with this application that you will encounter and that you should think about.

You are free to change the details, as long as the main functions described here are supported. In case of drastic changes, you might want to talk to the TA or me first. The project description is intentionally somewhat vague; you are free to decide what kind of interfaces to support etc.

2 Data

A brief description of the data that needs to be stored in the database follows.

- **People:** Information about the people who have agreed to be monitored. This includes the information about the name, home location (see below), work location etc.

- **GPS Information:** The GPS location information for the people in the database, along with the times at which the readings were taken. Assume that the granularity of the “time” is minutes. Note that this does not necessarily mean the information is available for every minute\(^1\). Also, historical information needs to be stored and can’t be discarded.

\(^1\)You may choose to make that assumption to simplify the rest of the tasks
• **Friends:** Information about peoples’ friends. Assume this is a symmetric relationship.

• **Locations:** A location (e.g. a cafe) is a rectangular region, specified using the coordinates (same as the GPS information). Some auxiliary information about locations (e.g. whether it is a cafe or a parking lot etc) should also be stored. This information could be stored as free text, or using a set of categories. Home and work for each person are locations as well (different for each person). Each location has a unique name of some sort (that could be an address). Along with that, a set of common locations are also stored (e.g. a cafe). Locations may be “nested”.

3  Tasks & Queries

There are two sets of tasks/queries that you need to support. The main tasks are provided below. You may choose to add and/or substitute other tasks depending on the emphasis of your project. For most of the tasks, you should provide a web-based interface.

The first set of tasks relates to the information that is stored in the database.

• **Insert a new location:** Insert information about a new location. Along with GPS coordinates and other fixed information (e.g. type of location, street address, web url etc), it should be possible to enter arbitrary free text about the location.

• **Register for the monitoring service:** A new user can register to use the service. Some information about the user would be collected while registering, and the user also has to choose a username and password to log in. Along with the password, a “secret passphrase” should also be stored per user (see below for the use of this).

• **GPS location updates:** We will assume that the location updates arrive at the system in XML format. Here is an example of an update:

\[
<\text{location-update}>
  \text{<username> X </username>}
  \text{<latitude> X </latitude>}
  \text{<longitude> X </longitude>}
  \text{<time> X </time>}
</\text{location-update}>
\]

Write a (stand-alone) program that will take as input one such update and insert the information into the database. A web-based interface is not appropriate for this task since it needs to be executed at a very high rate.
The second set of tasks relate to the user interaction with the system. Here are some of the tasks which should be provided.

- **Add friends:** Users can add someone as their friend if they know the username and the “secret passphrase” for that user (this is a crude way to “authenticate”; if you want to set up a more elaborate authentication scheme, that would be fine too).

- **Where are my friends?** The output should be a list of the friends along with locations, “unknown” (if location information is not available), or “outdoors” (if the current location does not fall inside any known location). If multiple locations match (because of “nesting”), show all of them. There should be a “refresh” button on the results page, to manually refresh the answer.

- **Browse historical data:** Given a period in the past, list the locations (names) the user was at and the fraction of time the user spent in each of the locations. Make (and state clearly) assumptions about missing location data.

- **Search through locations:** Given some information about a location, list all the locations that match.

- **Monitor the number of people at a given location:** Given a location (identified by some form of unique id), show the number of people currently inside that location. There should be a “refresh” button on the results page, to manually refresh the answer.

- **Look for abnormal patterns:** For example, find pairs of people who seem to have met often, and far from either person’s home or work location, during last week/month. The various parameters of the query (how far from the home/work locations etc) should be taken as input from the user.

**Populating the tables:** Populate the tables manually with sufficient tuples (10-100 each) to demonstrate the functionality of the system. Feel free to use made-up geographic coordinates etc. Actual geographic information can also be obtained from many sources. For example, Wikimapia (http://wikimapia.org) is a good source for such information. You might be able to find real GPS traces from the web as well.

**Extra credit:** There are many directions that you can explore to enhance your project for extra credit (upto 20% of the project grade/4% of the total grade). Some suggestions:

- **Sophisticated exposure of the location information:** Users might want a more fine-grained control over whom/where/when their location information is disclosed to their friends. For instance, it might be okay for the spouse to see which room in the house a user is, but a friend should probably only see that the user is at home. Designing an intuitive and usable interface for this would be challenging.
• **Fill up the missing data:** The collected data is unlikely to be uniform in time (GPS tends to be unreliable in cities etc; also data might be lost on the way to the central system). A simple way to take care of this is to interpolate by looking at the last known GPS update. So if a user was at location \( X \) at time \( t_1 \) and is at location \( Y \) at time \( t_2 \), we can use linear interpolation to fill up the missing values in between (this would be an okay approximation; in real life, more sophisticated techniques must be used). One way to do this is when a new location update is received, the (stand-alone) program that handles it would first find out the previous known location for the user, interpolate and insert all the values in between.

Be creative.

4 **Rules of the game**

• **Groups:** The project is to be done in groups of 2 students. A roster for each group must be submitted to the TA by the date specified in the “Due Dates” section of class schedule. The groups are “self-policing” (e.g., each group is responsible for its own division of labor, scheduling, etc.). *Note: If an unreconcilable problem arises in your group, it is your responsibility to contact both the professor and the TA as soon as possible. After the project is due, it will be too late.*

• **Assumptions:** In cases where you have questions on the above description, it is acceptable to make assumptions about the application providing that: 1) they are explicitly stated in the report, 2) they don’t terribly conflict with any of the requirements specified above, and 3) they are “reasonable”. If you have a question about the acceptability of any of your assumptions, check with the TA or the professor.

• **Reports:** A report should be handed in at the end of each phase (Due dates below).

• **Implementation:** The final phase of the project requires a working implementation of the system to be built, tested, and demonstrated. A large part of the project grade depends on the quality of this implementation. The implementation will be done as a client-server system in which a web server runs on your cluster unix account, accepts web queries, and connects to the Oracle DBMS to retrieve from the database.

5 **Project Phases**

<table>
<thead>
<tr>
<th>Phase</th>
<th>Phase Name</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Group names to the TA</td>
<td>Sept 19, 2006</td>
</tr>
<tr>
<td>II</td>
<td>Implementation, Testing, and Demo</td>
<td>Nov 30, 2006</td>
</tr>
</tbody>
</table>
6  Reports

The Phase I report must contain:

1. a short description of the purpose of the project and the purpose of this phase of the project.
2. a description of the scope of the project
3. the assumptions that you have made about the enterprise.
4. the graphical schema using the E-R model,
5. list of the attributes for each entity and relationship,
6. the relational schema obtained by mapping the E-R to relations, and their Boyce-Codd or 3rd Normal Form with keys.
7. a description of how you plan to populate the tables (including details of how you plan to parse the XML updates).

The Phase II report must contain:

1. Phase I report with corrections addressing TA’s feedback.
2. a description of the purpose of this phase of the project,
3. a description of the problems encountered in this phase and justification for the solutions.
4. any revisions made to the relational schema definition from Phase II,
5. a brief summary of your implementation efforts, the tools used etc.
6. a description of the system’s limitations and the possibilities for improvements.
7. In addition, a demo of the system is required. All members of the group should attend this demo, to explain the aspects of the project for which they were responsible.