Free University of Bozen-Bolzano

MOBILE SERVICES

Project 10: TRACKER

Raonne Barbosa Vargas, Adilson Oliveira Cruz, Martin Cetkovsky

1. Introduction

This paper is a technical report of the Location Tracker project of the Mobile Services lecture. In the following sections will describe the systems functions, how the human-computer interaction process happens, the structure of the code and the major technical problems founded.

1.1. Project Definition

The purpose of this is to implement a mobile J2ME application which has the following required functionalities:

- Record the position during a walk (interval or space interval).
- Save locally the collection of positions with a name.
- Load a collection locally saved.
- Export a collection in Google Earth KML to an external server.

In order to improve the project, some functionalities were improved and new ones were implemented:

- Pause / Resume support: while recording the positions the user can pause and resume at will, allowing more control over the recordings.
- "Continue Recording": it's possible to load collections and continue recording and appending new positions for it.
- Options and default values: the system always starts with the options chosen for the last time. It's also possible to pre-fill the default values for the next recorded locations, as setting the options to record the locations, and this will be pre-filled in the forms, or the last used address might be used.
- Special care about the user interface: the user interface is defined in a way to make easy to the user to quickly and easily use all available functionalities, like in user error management, according to the issues when using mobile devices.
- Delete Collection of Positions (that it was previously saved locally).

1.2. Solution

The project is implemented in a form of two parts - a J2ME mobile application (Midlet) which provides the main functionality and a J2EE server script (Servlet) to save the exported locations. In the mobile application the Location API is used to gather the current location, the Http communication and Servlet API are used to communicate with the servlet and the RMS API to store the locally saved locations and the user preferences.

1.3. Organization of the paper

The rest of the paper is organized as follows. First, the user instruction is presented. Second, the code and class structure is described. Finally, the major technical difficulties and its solution are discussed.

2. User manual

2.1. Main Screen

The main screen is reached after you start running the application or when you finish an execution of the main functionalities in the application. In the main screen, you can start a new recording of positions, setting an interval to capture the coordinates by time or by distance. The frequency on the example is to record an actual position each 10 seconds. To start the recording, choose the **Record** bottom (on Figure 1).
On the main screen you can also Load a previously saved collection of recorded positions. Selecting the option Load, you can see the collections, and choose the desired one, as shown in the Figure 2. Push **Load** to load this collection (Figure 3), and go to the Load screen.
The third option in the main screen is to go to settings of the application by pushing **Options** (Figure 4). Finally, you have also the button **Exit** to close the application.

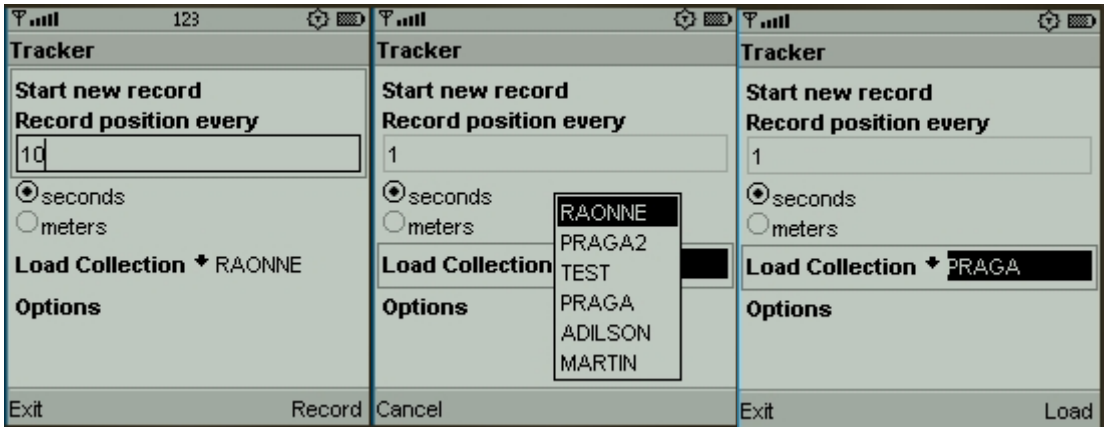

Figure 1 Figure 2 Figure 3

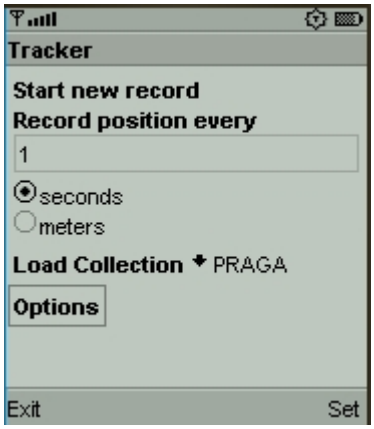

Figure 4

2.2. Start new recording

After selecting the **Record** option in the main screen, the Recording Position screen is shown (on Figure 5). Whenever a new location is retrieved, the list of the last recorded locations is updated. You might be asked for approval of retrieving the position, depending on the security settings of your mobile phone. You can cancel the recording without saving at any time using the **Cancel** command. If you wish to pause the recording for a while, you can use the **Pause** command (and then **Resume** to continue). After you are done with the recording, you can save it for a later time or export it to a server. To proceed to these options, choose the **Save** command from the menu.

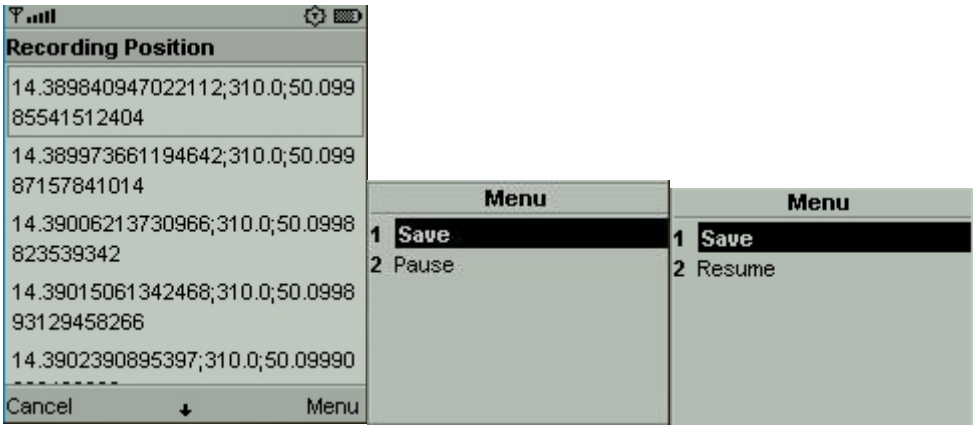

Figure 5

2.3. Save and/or Export

On the Record screen, if you select **Save**, you are sent to the Save Collection screen (Figure 6). Please note, that the recording is automatically paused now. Write the name for the recorded collection. Optionally, you can choose to export the collection in the Google Earth KML format to the specified server. To do that, check the **Export** checkbox. To save the collection (and export, if checked) use the **Save & Finish** command (if that name has already existed, it is automatically overwritten). Alternatively, you can save the collection and continue with the current recording (**Save & Resume**), or continue the recording without saving yet (**Resume**).

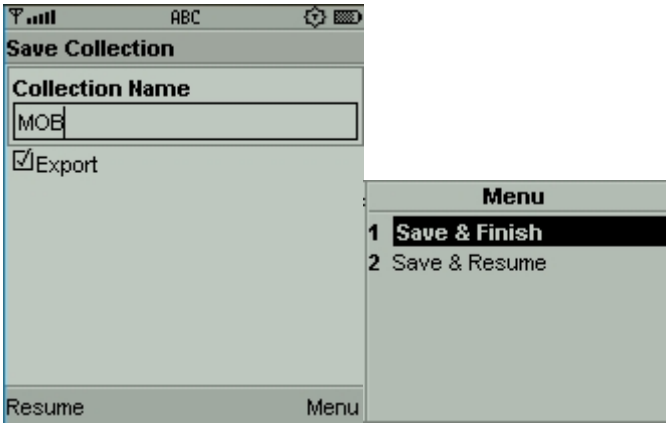

Figure 6

2.4. Loading a collection

The list of previously saved collection is available on the main screen under the Load Collection. To load a collection, simply choose its name from the list and click Load. The list of coordinates is shown. Once in the Load Screen (Figure 7), you can see the recorded positions on this collection. If you would like to continue recording, choose **Continue Recording**. You can also Export this collection selecting the option **Export**. Otherwise, choose **Back** to return to the main screen, or **Delete** to delete this particular collection.

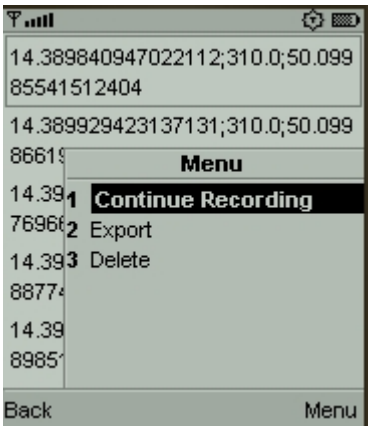

Figure 7

2.5. Options

The application supports saving default values for several fields - the frequency of a new recording and fields on the Save Collection screen. To change the settings, go to the Options screen (Figure 8) using the **Options** menu on the main screen.

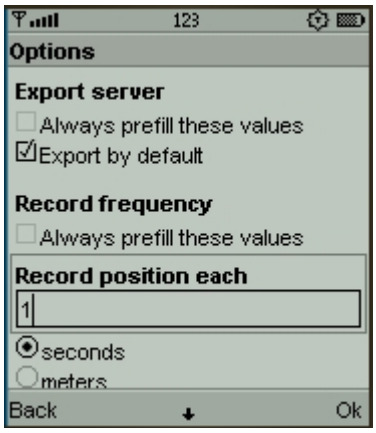

Figure 8

2.6. Error Handling

Considering that services such as connecting to the GPS System, and connecting to the Server are subject to unavailability, the application deals with this situations gracefully and offers the user the best options on how to proceed. Figure 9 shows the unavailability of the GPS System when the user is recording positions, and he has the option of **Save** the positions recorded until now (and even Export it) and to select **Ok** and just return to the Main Screen. Figure 10 shows the Connection to the server unavailable. The collection is not exported, but it's still saved locally and can be exported by the user in the future.

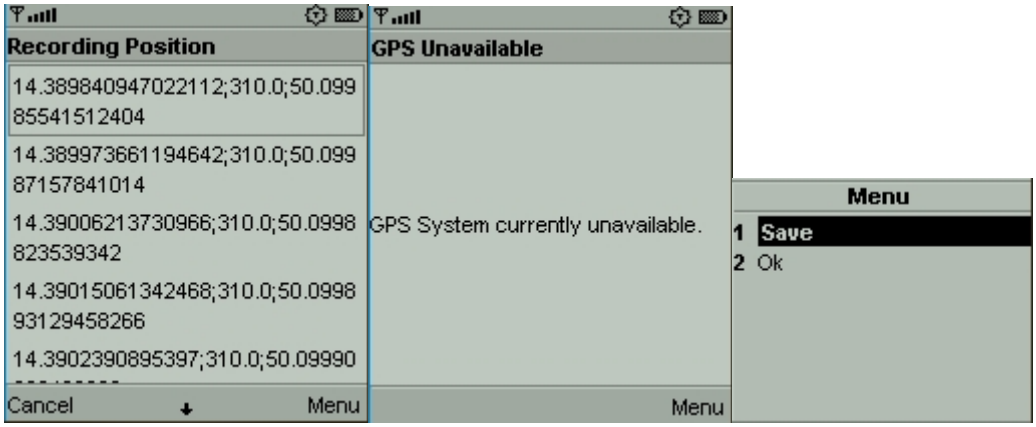

Figure 9

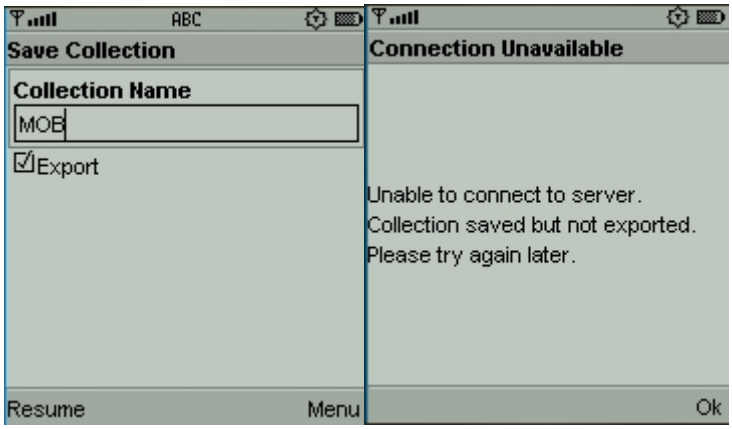

Figure 10

Other errors like bad input from the user (empty string for Collection name, negative number for time or space interval, and so on) are also handled, and a message is shown to the user to fill the form property (Collection name), or to use a default value (time or space interval).

3. Code structure

There are six important classes used in the project. The midlet is the TrackerMidlet class, which represents the user interface and top-level program logic. The Collection class is used to represent and store all the locations and collections. The GpsLocation class represents the thread gathering the locations from the Location API. The GoogleKmlFormat class creates the KML file content from the coordinates in the collection. The ExportKml class deals with connecting to the servlet and uploading the KML file. Finally, the TrackerServlet class is the servlet running on Tomcat that will receive the KML file and store on the server.

The TrackerMidlet class is the root class of the project. It uses all other classes (except the TrackerServlet) to perform the task requested by the user. The class itself does not process any function, only call the helper classes and handle the user interface.

The Collection class contains two main set of methods - to work with the list of saved collections (create, open, delete, save, getList, ...) and to update the coordinates in the collections (add, list) together with saving the location setting for case of resuming (interval of recording).

The GpsLocation class uses a separated thread and is used as a listener for the Location API. It inserts new locations into the Collection class and updates the list of last recorded locations.

The GoogleKmlFormat class receives the list of coordinates in the collection and the name of the collection, and generates the xml code for the Google Earth KML file, that will be exported to the server.

The ExportKml class also uses an independent thread, to deal with communication with the server. It uses a HTTP request POST to upload the KML file content to the server.

The TrackerServlet is compiled an executed separately, and has to be integrated in Tomcat to enable a server to listen and receive the uploaded KML file.

The application takes special care of the connections with the GPS and with the Server. Besides running in independent threads, both also have a graceful error handling, in which the user is always informed when a service is unavailable, and receives all the appropriate options on how to move forward.

The entire code was developed for this project by the three members of the group only. Some ideas in the code were of course inspired by the lecture slides with the samples for dealing with Hash Tables, the Location API, and the Servlet API and HTTP Connection. No code was re-used, but the theory and ideas behind them were obviously based on the material learned in class.

4. Major technical difficulties

There were several technical difficulties solved during the project development, usually due to the limitations from the devices and the lack of support from the API. The difficulties presented in this report are on the following topics: Retrieving location each x meters; Thread management; Uploading files to server.

The first difficulty comes from the fact, that the Location API does not support trigger with a condition like "let me know when the user leaves out of the circle of radius x meters around here". As a consequence, the active waiting method is used in this project - the current location is retrieved on the time basic, but recorded only when the distance from the last recorded position is more than the given amount of meters. As a result, the location service is used even the location is less than required and the time to request the location is according to the guidelines.

The API also does not support some basic functions for threads, like methods to pause, resume or kill. In order to overcome this problems, in the class that implements the Thread a boolean property called isPaused was created (accessed by the methods pause() and resume()), which is checked before executing the main tasks and a boolean property isAlive is checked in order to continue to execute the Thread. The location calls are no longer registered and used while these boolean holds appropriately.

To upload the KML files to the server was necessary to set up Tomcat with a servlet to process the requests from ExportKml and properly store the KML files on the server. Tomcat requires a very strict structure for the positioning of the servlet class, the definition of the URL (that was also set as a property of the midlet, as recommended), and the creation and maintenance of XML files related to its setup. With a lot of effort, all difficulties were overcome and communication is working perfectly. Instead of posting a file through the HTTP connection, the content of the KML file is sent as a

string and the servlet is in charge of storing it in a appropriate file on the server, using the name of the collection as the name of the KML file.

5. Summary

All the required functionalities as well as several bonus functions are implemented and it consists of two parts - a mobile application (Midlet) and a server script (Servlet). The project uses several techniques and libraries to perform its task - for example J2ME, Location API, RMI and Servlet API. Additionally, the project was tested in the WTK 2.5.1 environment (Midlet) and under the Apache Tomcat 6.0.13.