Abstract

This report is about a J2ME project developed by Māris Bungše and Federico Leverato for the Internet and Mobile Services course during winter 2010/11.

Goal of the application was to assist a stranger in finding nearby places of interest in the Trentino region, using a given source for the geotagged categories and places. The development led to a solution providing the possibility to use the same idea both for „on-the-road“-use and for planning trips or visits while not in the area of interest.

Also technical problems and suggestions emerged during the development are reported.

Typical user:

Independent not-guided tourist or occasional business man or anyone else in similar mood/situation.

1. They’re traveling, not familiar with the surroundings and they don’t have a direct way to get information (or are not willing to use the offered ones)
2. Rather wealthy, may have basically two distinct goals:
   a. look for sightseeing opportunities, cultural or other events as well as museums or historical sites.
   b. in a hurry, needing a place to eat or where to stay (primary needs)

_Contextual conditions:

1. **Physical context** – one or more people together are traveling by car or train, walking through an unknown city or sitting in their hotel room. It may be limited knowledge of local language or there may be other obstacles to get information or recommendations by other people or sources.

2. **Media context** – they are using their phone and mobile internet to find out what is around them. They use average to high end mobile phones with GPS functionality and have a "young" attitude for technology and new uses thereof - meaning they don't fear trying and experimenting e.g. new phone apps or functionalities. They will consider using their phone to help them out of an unpleasant or unfamiliar situation, actually preferring it to other ways of getting help.

3. **Modal context** – users can be very excited as well as tired, but they know very well how to use their device. They want to find an accommodation or attractions and restaurants near them. They trust the application's suggestions.

_Context based categories:

1. **Locale**
   - This is a typical case of a locale based application. It's use is strongly connected to the presence of a GPS chip on the phone, although it might be used also without when e.g. planning a trip.
1. **User Functionalities**

a. The user starting the AroundME application may directly choose a category between the proposed ones: accommodations, events, eat&drink, must-see and sport.

b. That leads to a listing of matches depending on the actual settings and the coordinates used, where the user may choose among the suggested entries. These are sorted from nearest to farthest.

c. The third page is the detail page; the user may see a picture as well as read the most important details about the chosen item.

d. The user may choose to view the given location on the map, using an image which is downloaded from Google's Map servers.

e. He may also choose to create an event in the phone's calendar (PIM) with the item's data - as a reminder or for planning purpose.

f. He may instead choose to go to the settings screen through the soft menu on the main menu page.

g. The user may choose the maximum distance (from himself/his device's position or from a chosen pair of coordinates) for the shown results.

h. He may choose to switch between the use of GPS and manually inserted coordinates (still limited to Trentino's area).

i. The first time the user opens the application, he needs to go to the download function in the settings screen: AroundME will download the data within a specified radius (see i. and f.) to the device for later use.

j. A criterion similar to f. may be used for the downloaded data.

k. Settings may be saved or changes cancelled when returning back to the main screen.
The following use case shows the two main uses of the application:

2. **Architecture**

The application was developed using the following development resources:

<table>
<thead>
<tr>
<th>Developer Toolset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java ME SDK 3.0 (mac)</td>
<td>J2ME IDE</td>
</tr>
<tr>
<td>Netbeans 6.9.1 (win)</td>
<td>J2ME IDE</td>
</tr>
<tr>
<td>Eclipse Galileo and Helios with eclipsme plugin (mac)</td>
<td>J2ME IDE</td>
</tr>
</tbody>
</table>

See part 4. and 5. for further details and an explanation about this *multitude*.
Moreover, the following libraries were used:

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kXML version 2 (xmlpull 1.1.3.1)</td>
<td>Lightweight XML Pull-parser API (*)</td>
</tr>
<tr>
<td>LWUIT version 1.4</td>
<td>A graphical user interface library for J2ME,</td>
</tr>
<tr>
<td>Microfloat / Microdouble</td>
<td>Implementations of float and double for</td>
</tr>
<tr>
<td>Google maps API</td>
<td>API to retrieve map sections, here as static</td>
</tr>
<tr>
<td>JUnit (different versions)</td>
<td>Unit testing</td>
</tr>
</tbody>
</table>

(*) kXML was used directly from source; some unused parts were removed, some methods where slightly adapted to the project's needs. It is itself based on the xmlpull library which was used as jar file.

CLDC 1.1 and MIDP 2.0 were chosen as target, occasionally switching to MIDP 2.1.

The project was proposed and tutored by ectrlsolutions.com, a software solutions agency based in Trento, which also provided the source of geotagged data. The data source was a web server accessed through a RESTful web service requiring different configuration and authentication steps before allowing somebody to query - similar to the hand-shake of some network protocols:

- **Session opening**
- **Language choice**
- **Authentication**

After successful completion of the three mentioned steps, a query may be sent to the server:

- **Querying**

This last step may be repeated until the http session expires, which requires restarting the whole procedure from the start.
The following is a scheme of the main parts of the application’s architecture:

As may be noticed in the scheme, the application uses different technologies and features to achieve all its tasks:

- HTTP-Connections to communicate and retrieve informations from ectrlsolutions.com and from google maps. For these connections, a WIFI, GSM or 3G data connection is needed.
- The GPS chip to get the current coordinates and find nearby places
- A special API to store and retrieve files locally as well as to store and retrieve events from the device’s PIM (personal information manager, which usually handles a calendar with events and an address book with contacts)
2. **User Interface**

The following AroundMe user interface forms illustrate the functionality of developed midlet application:

1. **mainForm** – also the welcome form

2. **listForm** – showed after searching the Landmark store
3. detailsForm – shows details of accommodation

4. mapForm – visual representation of current position and place of interest
5. Development strategies

Before deciding any further strategies, the data source had to be analyzed in detail. Being a pre-existing system, not custom made for this application, seemed the biggest challenge from the beginning!

See 5. for more details.

During development we had to face various tasks:

1. Retrieving data from the source
2. Parsing xml
3. Store information locally
4. Decide whether to work online or offline (as a user)
5. Organizing the UI

Point 1. seemed trivial. Point 2, 3 and 5 resulted to be the toughest.

Point tree was needed in more than one situations. We finally came up with this strategy:

- Store the xml files containing the data locally, in the application’s private folder
- Store the data parsed from xml into an extended form of Landmarks and then a Landmarkstore.

(The idea was to extend the Landmark class in order to have more information stored, using a mechanism which was able to build strings out of chains of information which then were added to the description fields before storing and extracted when reading. It turned out to be a bad idea...)}
- Store the users settings using a Recordstore
- Some fixed resources like the icons came as internal application resources, stored in the res folder

5. Technical issues

Mac & J2ME
- We found out that it's not a good time to develop J2ME solutions on a mac. The current J2ME SDK 3.0, JMUnit and LWUIT are causing all kinds of errors: LWUIT is causing pre-verification errors which could not be avoided. The only apparent solution would have been to recompile LWUIT after removing the offending classes, but the attempt to do so also failed for compile-errors.
- Moreover, we initially were trying to do test driven development. After a lot of trying and experimenting, also with changed and modified versions of JMUnit, we found the current versions to be not working with the current SDK: the reason was it tries to create widgets from within a widget which is clearly in contrast with J2ME's rules. Without testing development became much more complicated and basically slower.
- Also switching from SDK to NetBeans or even Eclipse did not bring to better results on the mac.

Only Solution was to switch to Windows for the development (after a lot of wasted time)

Landmarks
- We found the J2ME Landmark/Landmarkstore to be interesting. It seemed to be good for our goals. Bad thing was the lack of a clear and uniform definition of its limitations: what can be stored in a Landmark? The name has a 32 Bytes limitation, but what about the other fields?
We decided to extend the Landmark class with our AMLandmark class, which was adding an ExtraInfo Object containing the things we needed. To make it work with a Landmarkstore, we used a trick: ExtraInfo had toString() and fromString() methods which chained all information together in a string and added it to the Landmark's description field when storing. When reading the Landmark back from the store, it was a matter of extracting the ExtraInfo from the field before using everything as usual.
Very unluckily, this theory revealed to be false at the very end of our development: although the Landmarks could be stored and retrieved without any error, the retrieved versions were all missing the whole description content, containing null instead.

The solution was to store the ExtraInfo strings in text files: one for each record and using the Landmark's description field as a connection.

The xml files
- The source of our data were files which compiled the item's informations used for a website into xml files. These xml files were not ideal for a J2ME solution. They had an overcomplicated, sometimes inconsistent and unsorted design, making it
time consuming and still error-prone to fetch all the needed data. Moreover, each
category had its own set of tags and was to be handled separately - but that's
probably due to the nature of different items and therefore unavoidable.
We brought suggestions to make those files more efficient. See 6.

The solution was to leave out part of the available informations. A tag-mapping
class was used to map different sets of tags to different categories.

6. Future development

The application has lots of possible future development. First of all, it should be made
more robust: a consistent error handling and showing capability would be useful, as well
as a log file. All settings should be cross-checked to avoid possible errors or inconsistent
situations. Functions should activate/deactivate according to these checks. Some details
of the GUI could maybe be still smoothed out and simplified. That also depends on proper
testing „on-the-road“ and changing according to feedbacks.
But the most important part to work on for future development is the data source: the xml
structure may be optimized to make parsing easier and faster. And, if possible, the quality
of the information inserted (probably from the website’s clients) should be taken care of
from the beginning: a more strict control of input, a better handling of missing or default
information, a nicer formatting (All upper case, _ or - instead of the empty space, etc.).

Cases like:

<keyword>
<kwID label="Ristorante">RESTAURANT</kwID>
</keyword>

which are frequent, may be changed to

<keyword kwID="Ristorante">RESTAURANT</keyword>

making it easier to parse, although there’s still a language inconsistence and the upper
case issue in this case which might also be worked on.
A structure like

<record>
  <recordfield1">RESTAURANT</recordfield1>
  <recordfield2">Pizzeria Whatever</recordfield2>
  ...
  ...
  <recordfieldX">Trento</recordfieldX>
</keyword>

may be ideal for a J2ME application.