Domotics
Internet and Mobile Services

Patrick Clara, Manuel Piubelli

2010/11

Abstract

People do more and more like “less important” things to be done by themselves, possibly avoiding to even think about these problems. The management of the house in terms of switching lights, heating, watering, in terms of opening and closing doors, windows and shutters and in terms of keeping the right temperature according to the actual needs is certainly one of the things people would like to be done by someone else and are considered to be annoying. Domotics, a J2ME mobile application provides a way to accomplish these tasks by a few keyboard hits from any place using the mobile phone. Moreover, Domotics implements an intelligence module able to manage the house automatically using contextual information as the user position and his movements.

This report presents the features of the application but does also describe the analysis, the structure, the human/computer interaction and a final conclusion on the project. The Domotics project was developed for the Internet and Mobile Services Course at the Free University of Bolzano taught by Prof. Francesco Ricci.
## Contents

1 Project Analysis ................................................. 3
   1.1 User and Context Analysis .......................... 3
   1.2 Use Case Diagram and Sample Scenarios .............. 3

2 Functionality .................................................. 5
   2.1 Core Features ............................................. 5
   2.2 Intelligence ................................................ 5
   2.3 Additional Features ....................................... 6

3 Human/Computer Interaction ................................ 7
   3.1 Structure .................................................... 7
   3.2 Features ...................................................... 8
   3.3 LWUIT .......................................................... 9
   3.4 Server Interface ............................................ 9

4 Application Structure .......................................... 10
   4.1 Server Structure ........................................... 10
   4.2 Client Structure ........................................... 10
   4.3 Interaction .................................................. 11
   4.4 Technologies used ......................................... 12

5 Code Structure .................................................. 13

6 Conclusions ...................................................... 14
   6.1 Technical Issues, Problems and Solutions .......... 14
   6.2 Possible Improvements .................................... 15
   6.3 Project in Numbers ........................................ 15
   6.4 Lessons Learned ............................................ 15
1 Project Analysis

1.1 User and Context Analysis

User: The user of our system is probably wealthy/affluent people, since she/he owns rather big houses, apartments or villas where such a system is more useful and since they afford to buy expensive hardware for this system. The typical user will probably also need a minimum of familiarity with electronic devices.

Device: Since the users are probably wealthy, they do probably own expensive cell phones or business phones (like blackberry) with large screen space. However, our application is designed to work on every J2ME compatible phone and does only require having SMS and, for intelligence mode GPS enabled. A colour screen with some good resolution is advisable for a better user experience.

External Context: Information about position, movements and time influences the intelligence part of the system, especially heating, alarm and watering are dependent from the external context. Also interaction changes when user is away (not near home), since in this cases alerts from alarm etc. are sent in interrupting mode. Whereas this is not done when the user is at home.

Software type: This software should be a utility application since users are using it mostly when they are away and expect it to be effective and fast. For this reason the application should have an easy and intuitive interface

1.2 Use Case Diagram and Sample Scenarios

The Use Case Diagram (Figure 1) depicts the most general features of the application, whereas the sample scenarios below describe more precisely one detailed usage situation each.

![Use Case Diagram](image)

Figure 1: Use Case Diagram

S1 - User is at home and wants to close the shutters and turn on the lights while he is lying on the sofa, he accesses the application and selects to control the respective room with 3 menu selections (details -> room -> light2) and then turns the light to “ON” and clicks on
shutters 2 setting them to “DOWN”. When finished selecting he clicks on Menu - > Send Changes, which notifies the server and thus all other client are notified of the new changes.

S2 - The user is at work and receives an alert that someone broke into his house by a message, the police is notified by the user, and the user can drive home immediately to fix things with the police. (we suppose the alarm system automatically notifies the police).

S3 - The user is returning home from a journey and has intelligence mode switched on, the system detects that the user is approaching home by analysing the GPS coordinates and notifies the server. The house is therefore automatically heated and the user has a warm comeback.

S4 - The user is on vacation and it is very dry, remotely, he can water his garden every day by a simple instruction. S5 – The user leaves the house for work, the system checks whether other users are in the house and it locks the house otherwise.
2 Functionality

2.1 Core Features

Our application, intuitively described, is able to check the states of all devices of the house, according to their location and type and to change the states of these devices remotely. In order to create realistic and usable software, we structured it in an abstract manner such that any device could be included in a house, in fact also new device types can be added with few lines of code. In our sample house, we have the following devices:

Room

- Main Door
- Heating
- Lights
- Shutters
- Windows

General (House)

- Alarm
- Fire Alarm
- Garage Door
- Garden Watering
- Garden Door

Each of this devices may be switched on and off or, respectively closed and opened, moreover, for the heating, the user can specify the desired temperature. If the temperature is different from the current, the heating is also automatically switched on.

2.2 Intelligence

In addition to the “traditional” functionalities of accessing and controlling the house from remote in a manual way, our application provides some more sophisticated, intelligent feature. Since users do like the maximum of commodity and avoid the discomfort of controlling the devices manually, we implemented the automatic controlling of the house according to the users position and movement. The system virtually divides the user’s world in three concentric, circle-shaped areas with certain distances from the house as shown in Figure 2. When the user moves from one circle to another, say, circle 0 to circle 1, the house reacts on this contextual condition by changing the houses state according to the users needs (e.g. it locks the door or switches off the light). The distances of the circles from the house are easily specifiable and should depend on the users habits (i.e. how far he lives from his workplace and where he spends his time).
The intelligence of the systems does also take into account the position of other users, this is an important feature since missing this would mean to create misbehaviour (e.g. the house switches off the heating while a user is still in the house).

2.3 Additional Features

The application defines a set of additional features improving the quality of our application in terms of communication and reliability:

Reliable Alert Sending: When the server sends an alert, the communication is reliable, which means that the server keeps sending the alert until the client does notify the reception of the Alert.

Client Authentication Only known clients, i.e. clients and phones that are registered in the server are allowed to control the house, such that no stranger can for instance open the main door of the house.

Features related to the user interface are described in section 3.2.
3 Human/Computer Interaction

The Graphical Interface of the client gives a qualitative visual experience to the user. Through the usage of the LWUIT interface API, we could reach a high level of personalization. Our application, rich of images and of different items, provides a simple, fast and intuitive way to retrieve and to input information.

3.1 Structure

The client user interface provides a classical navigational interaction such that the user may immediately access the most frequently used commands or information. The first screen in fact allows the user to access the most important and general commands. Increasing the level of detail, the user has to browse through more forms. A further advantage of our interface is the correspondence between the natural nesting of a house in rooms and of a room in devices and our browsing experience. Once a user has selected a device in the location-specific or in the general listing, a dialog, which allows the user to change the status of the selected device, appears.

![Figure 3: Browsing](image)

Figure 3 shows a typical browsing to the lowest level of detail (e.g. to switch the specific light of a specific location on and off). In addition to the forms providing information regarding the status of the devices, the client interface features a settings-form, which permits the user to set the server address, to specify where the house is and to switch between intelligent and manual mode.

Since we decided to create a utility application, the interface is designed for short and target-driven usage. This, together with the motivation of having the interface always up to date led us to the decision of creating the forms “just in time” i.e. when they are needed. This may leads to the re-creation of an already existing form, but is much more efficient with respect to pre-creating all forms for short browsing (e.g. for a probable amount two modifications our method needs to create from 6 to 7 forms, whereas the pre-create strategy will create almost 20 forms, depending on the amount of locations in the house. Obviously, for longer or continuous browsing, pre-constructing would be more convenient.
3.2 Features

Besides the functional and basic functionalities of browsing and modifying the settings our user interface provides a number of special features, including: Status icons. Each device in a list has its own status icon, showing whether the device is on/off open/close etc. This feature allows the user to check the states without entering in to the device-specific screen. Moreover, the icons are self-created and personalized according to the device type.

Delayed sending of instructions: In order to avoid human mistakes from the user and in order to minimize interaction with the server (providing smaller costs), the client does only send instructions of changing the status of a device when the user tells the client to do so explicitly via a command. Up to that point in time, all changes are stored locally. When the user decides to send the modifications made, they are sent cumulatively.

Strictly related to the previous feature, when a user decides to save a modification, first a check ensures that the user really changed the status (i.e. switching on and off the light is no change), and, if so, a special character * near the status icon appears. This character disappears when the changes have been sent to the server (i.e. they have been applied). This allows the user to check the modifications he ordered up to any point in time and is shown in Figure 4.

Finally, if the user forgets to send the saved changes and tries to exit the application, a dialog asks him whether he wants to commit the changes to the server or if he simply wants to exit without committing. This feature is also illustrated in Figure 4.

![Figure 4: Changes](image-url)
3.3 LWUIT

The Lightweight User Interface Toolkit provides an easy and effective way of high personalization for a mobile application. The classes have an amazing variety of features, which can be used to satisfy the needs of mobile developers. Moreover, LWUIT comes with a theme editor, which allows programmers to design the mobile interface in a css like fashion, having style and functionality completely separated. In our application, we created our own nice and clean theme, which gives an additional flavour to the interface and won’t add any complexity the code through style instructions.

Unfortunately, LWUIT is not yet bug-free, the rendering of the borders on the menu is sometimes imperfect. Similar, unsolved issues were reported on the official LWUIT forum (see references).

3.4 Server Interface

The server of our Domotics application is supposed to be a running on a device directly connected to the electronic system of the house. It was not designed to be a fixed control station with user interaction. For such a purpose a client running on a CDC device would be the best solution. Our client is developed in a modular way and therefore it would not be difficult to apply another communication method (e.g. Ethernet) to it.

However, our server has a minimum of user interface for security and testing reasons. Through this interface, the user is able to monitor all devices and to set the clients addresses such that only authorized numbers can control. Also, when a devices status is changed remotely, an alert notifying the user is shown.

![Server Interface](image)

Figure 5: Server Interface
4 Application Structure

4.1 Server Structure

The Server is structured in a most modular way, consisting of the four following parts:

1. The receiver receives messages from the client side device and passes them to the Device controller or the Intelligence module depending on whether the content of the message is a change in position or an explicit client command.

2. The device controller is responsible for changing the device’s state according to the instructions coming from either the receiver, in case of user commands, or from the intelligence module.

3. The Intelligence module receives the client movements from the receiver, and according to where the user is headed, it issues commands to the device controller.

4. The status updater is a thread responsible for sending updates to all client devices whenever some device of the house changes its state (push). Moreover, it has to notify the user device whenever there is some alert (e.g., fire or security alarm).

![Server Architecture Diagram](image)

Figure 6: Server Architecture

Figure 6 represents the modules and interaction among them. For a better performance, the receiver and the updater are implemented in separate threads.

4.2 Client Structure

The Client Application consists of three major independent modules with the following responsibilities:

1. The Position Updater gathers the position of the user through GPS and checks whether the user has changed area in a cyclic manner. If so, a message with information about this movement is sent to the server. This module is registered in the PushRegistry with an alarm.
2. The Status Updater receives updates from the server when some devices in the house changes their status and stores the updates into a RecordStore. Moreover, the Status Updater receives alerts in alarming situations, displaying them to the user. Also this module is registered in the PushRegistry and activated when a message arrives.

3. The Interface makes up all user interaction. It reads the states of the devices, displays them to the user, lets the user change the states and sends commands to the server.

The Position and Status Updater should possibly run in background, whereas the user interface is the real interface and application. The latter benefits from the services of Status updater, reading always an updated status from the file.

![Client Architecture Diagram](image)

**Figure 7: Client Architecture**

### 4.3 Interaction

In the flow diagram shown in Figure 8 we represent the four manners of interaction between server and client, which are:

1. The client sends a message to the server when the user changes some devices status and applies his changes. The server will then change the devices status and notify all clients of this changement.

2. When the Position Locator detects a significant user movement, the server is notified via a message. The server intelligence module reacts accordingly, and the clients are notified of eventual changes.

3. In emergency case the server sends an alert message to the client. The user is then notified client side and an acknowledgement is sent to the server. As long as the message is not acknowledged, the server keeps sending it.

4. When some device changed its status for any reason, the server updates all clients of this changement.
4.4 Technologies used

**J2ME** All the code of the application is written in Java Mobile Edition. However, the interface is designed using the external Lightweight User Interface Toolkit (LWUIT), which is described in section 4.3.

**Location-API** The location API is used to retrieve the position of the user in order to allow the intelligence to react on position changes (movements) of the user.

**SMS** The interaction between the mobile application client and server is done through SMS, following our self-defined protocol described in section 3.3.

**PushRegistry** The Pushregistry is used in two circumstances in our application. In the former it activates the positionmidlet in a cyclic manner in order to send the movements of the users to the server. In the latter circumstance it is used to update the states of the devices of the house when the application is not running, and is implemented in an interrupt-driven way (the Updater is activated only when something really changed). This allows steady up to date information and no waiting for updates at each startup.

**RecordStores** We make extensive use of record stores to store important data in a persistent manner, more precisely, recordstores are used to store server-updates on client-side and application settings on both server and client.
5 Code Structure

The following table describes the responsibilities of each package and its classes.

<table>
<thead>
<tr>
<th>Package</th>
<th>Classes</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>- ServerMIDlet, UserInterface, ConnectionManager, SenderThread, ReceiverThread</td>
<td>The server package contains all functionalities related to the server from the interface to the intelligence. Apart from the Server MIDlet, which initializes all necessary objects needed for the connection and for the human interaction, the server package has a UserInterface, responsible for showing the states of the devices and changes to the user. Furthermore we have a ConnectionManager which starts and manages the incoming and outgoing connections. Sender and Receiver are implemented as threads responsible for the interaction with the clients.</td>
</tr>
<tr>
<td>Client</td>
<td>- ClientMIDlet, UserInterface, DeviceButton, Listener, SenderThread, ReceiverThread, ConnectionManager</td>
<td>The client package is responsible for the interaction with the user and for sending instructions specified by the user to the server. The ClientMIDlet is an initialization and cleanup - class, whereas the UserInterface contains all GUI - specific methods and fields. The DeviceButton is an extension of the LWUIT button, which allowed us to store and show the icons regarding the states of the devices. Furthermore there is a CommandListener “Listener” and a SenderThread which allows the client to send instructions to the server. The ReceiverThread receives alerts from the Server and the ConnectionManager manages the Sender- and ReceiverThread.</td>
</tr>
<tr>
<td>Updater</td>
<td>- UpdateMIDlet, UserInterface, ConnectionManager, ReceiverThread, SenderThread</td>
<td>The UpdateMIDlet instantiates objects needed for the updater, a small UserInterface is needed only to show that the application is updating the house’s states only in case this takes longer time than expected (usually the user doesn’t even notice). The ConnectionManager handles the incoming and outgoing connections. The receiver gets the updates from the server and the sender sends acknowledgements for important notifications by the server.</td>
</tr>
<tr>
<td>PositionSender</td>
<td>- PositionMIDlet, Locator, Sender</td>
<td>These two classes are responsible for sending eventual changes of position to the server. The Locator detects the user position, compares it to previous positions and checks if the user changed area (0-1-2), if the user did so, the server is notified through an SMS.</td>
</tr>
<tr>
<td>General</td>
<td>- House, Device, RecordStoreManager, Globals</td>
<td>The general package contains classes and structures common to both the server and client side, such as the data structure for the house (classes House and Device), the RecordstoreManager, creating an API for opening, closing, writing to and reading from records and finally a class of global, static variables called Globals.</td>
</tr>
</tbody>
</table>
6 Conclusions

6.1 Technical Issues, Problems and Solutions

Issue We experienced major problems in testing and debugging phase due to some emulator-related problems. For instance, an error message “MIDlet Suite is already running” may appear if the application is switched off and on again. This may happen because of the fact that our application uses the pushregistry to wake itself up on sms and on a certain interval, therefore it "stays alive" over different executions. When executing domestics a second time, the "old" domestics is still having the active pushregistry in the emulator, this will cause the error message. However, the original domotics application remains on the phone and works perfectly. This problem may only occur only appear in the client application when using the pushregistry, and does not appear in SDK 2.5.

Issue When running in standard execution, recordstores are not preserved as for applications run over OTA. For this reason, and since our application accesses and modifies the recordstores at each run, if we try to execute the application several times in standard execution it might happen that the recordstores are deleted since “marked to belong to a standard executed application”. This may happen after the first standard execution, and is a again emulator dependant. This problem does not appear in SDK 2.5 and will not in real mobile phones.

Issue If two PushRegistry-activated midlets are executed in the same moment the emulator in SDK 3.0 will crash. This is a very uncommon case and depends again on the emulator.

Problem Designing our user interface on paper was not too difficult, but implementing it using J2ME was much more complex and restrictive. With the instruments provided by the API we could not realize what we had in mind.

Solution We decided to use the leightweight API, as already described previously. This gave us much more freedome in composing our GUI.

Problem To keep all clients up to date, we would have to let the user wait for the update each time he/she starts the application. This would be very bad for a utility application, which should be fast and effective.

Solution Our application uses the pushregistry and therefore receives and stores updates from server side at any time. Furthermore, updates are only sent when needed.

Problem Our application does cyclic updates even when the user is not using the application in order to react on movements of the user and in order to keep the states of the devices up to date. Even if our application minimizes interruptions, sometimes the application interrupted itself for making updates/checking the location.
Solution  This problem was solved by deactivating the pushregistry before starting the client and activation of the pushregistry before terminating the client. Information is therefore always up to date and the server does live-updates regarding the position of the users (only when the user really changed position).

6.2 Possible Improvements

We tried to make our application as extensive and usable as possible. However, some improvements may still refine and increase the quality of the application. The most important ones are:

- One important improvement would be to let the user load xml files specifying the structure of the house into the system, such that the locations and devices of the house are not hard-coded. The integration with our application would be rather easy, since we have a very modular data structure for storing house - properties.

- The intelligence of the system is a bit rude up to now, a more dynamic system recognizing a users daily activities and drawing inferences from these observations would make our application yet more interesting. However, this is a very advanced topic and was not feasible for us in this time and with our knowledge.

- The user interface could be refined in some details (e.g. the * marking a changed device could be replaced by some nicer icon, which would not be difficult but time consuming.

- Updates could be received also at run time. This is not very difficult but also not very useful since the probability that two users are contemporarily controlling the house and using a utility application is very low and since the server checks conflicts in commands in any case. At the very next use, all states will be updated without any delay in starting the application anyway.

6.3 Project in Numbers

<table>
<thead>
<tr>
<th></th>
<th>Man Hours Est.</th>
<th>Man Hours Effect.</th>
<th>LOC</th>
<th>Classes</th>
<th>Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client (All Modules)</td>
<td>70</td>
<td>106</td>
<td>1297</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>Server</td>
<td>50</td>
<td>50</td>
<td>603</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>General</td>
<td>/</td>
<td>8</td>
<td>438</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Overall</td>
<td>120</td>
<td>164</td>
<td>2338</td>
<td>22</td>
<td>5</td>
</tr>
</tbody>
</table>

Finally, we committed exactly 135 revisions of the project on our SVN repository.

6.4 Lessons Learned

- Mobile Applications have to be designed starting from the interaction with the user, not like in traditional systems.

- Mobile Development is not like development for computers. Be it from the point of view of the user interface or of the available APIs be it from the point of view of testing and debugging an application, mobile development is much more restricted and limited with respect to “normal” application development.
• APIs have to be studied before they can be applied. We learned that a quick look at an API is not enough to be able to apply the API in an application, this may result in buggy code. A deep insight and study is necessary before using an API.

• The same application does not run equally on every device. The emulators of different SDKs gave different results and problems, this provided us big troubles in the debugging phase. We have learned that debugging on several different emulators and using different debuggers is more effective and time saving.

• A project will always take much more time than initially planned. Debugging and improving activities are time consuming and hard to estimate. However, the project was worth its effort and gave us a deep insight into the world of mobile development and satisfaction as it finally worked.
References

[1] Slides of Internet and Mobile Services Course 2010/11 by Francesco Ricci
   http://www.inf.unibz.it/~ricci/MS/

[2] Beginning J2ME by Sing Li and Jonathan Knudsen

   http://www.j2meforums.com/forum/

   http://www.java.net/forums/mobile-embedded/lwuit

[4] J2ME API
   http://download.oracle.com/javame/config/cldc/ref-impl/midp2.0/jsr118/index.html