

SPIDER

Social-Mobile Network

Antonio Sapuppo
{antoniosapuppo@gmail.com}

Supervisor:
Frank Fitzek

31th March 2008

Contents

| | | |
|------------|---|-----------|
| I | Introduction | 4 |
| 1 | Spider concepts | 5 |
| 1.1 | Introduction and Motivation | 5 |
| 1.2 | What is Spider? | 5 |
| 1.3 | Preliminary architecture proposal | 6 |
| 1.4 | Thesis overview | 7 |
| | | |
| II | Prestudy | 8 |
| 2 | Technology evaluation | 9 |
| 2.1 | Application server | 9 |
| 2.2 | Database | 9 |
| 2.3 | Wireless technology | 10 |
| 2.4 | Target Device | 10 |
| 2.5 | Development platform | 12 |
| 3 | In depth technology study | 13 |
| 3.1 | Sun Java Application Server | 13 |
| 3.1.1 | Sun Java Application Server architecture | 13 |
| 3.1.2 | Sun Java Application Server Security | 14 |
| 3.2 | MySQL | 15 |
| 3.2.1 | Mysql Architecture | 15 |
| 3.2.2 | MySQL Security | 16 |
| 3.3 | Bluetooth | 16 |
| 3.3.1 | Bluetooth Protocol Stack | 17 |
| 3.3.2 | Bluetooth Connection | 18 |
| 3.3.3 | Bluetooth Security | 19 |
| 3.4 | Java 2 Micro Edition | 19 |
| 3.4.1 | Java Architecture | 20 |
| 3.4.2 | MIDlet | 22 |
| 3.5 | Nokia N95 | 23 |
| | | |
| III | Implementation | 25 |
| 4 | Server side: Internet connectivity | 26 |
| 4.1 | Database Design | 26 |
| 4.1.1 | Logical schema | 26 |
| 4.1.2 | Strategy of the database implementation | 28 |
| 4.1.3 | Examples of using the strategy of the database implementation | 31 |
| 4.2 | JDBC | 32 |
| 4.3 | Servlet | 33 |
| 4.3.1 | Client requests | 33 |
| 4.3.2 | Server responses | 35 |

| | | |
|-----------|--|------------|
| 5 | Client side: Internet connectivity | 37 |
| 5.1 | RMS - Record Management System | 37 |
| 5.2 | Client actions | 38 |
| 5.2.1 | Initialization actions | 38 |
| 5.2.2 | Mission actions | 49 |
| 6 | My virtual world: Bluetooth connectivity | 68 |
| 6.1 | User Interface | 68 |
| 6.1.1 | Screen design | 68 |
| 6.1.2 | Virtual people and timers | 71 |
| 6.2 | P2P application | 73 |
| 6.2.1 | Server | 73 |
| 6.2.2 | Client | 74 |
| 6.3 | My virtual world actions | 77 |
| 6.3.1 | Client actions | 78 |
| 6.3.2 | Server responses | 91 |
| IV | Testing and Evaluation | 93 |
| 7 | Test and evaluation project | 94 |
| 7.1 | Unit tests | 94 |
| 7.2 | Usability tests | 98 |
| 7.2.1 | Initialization actions tests | 99 |
| 7.2.2 | Mission actions tests | 99 |
| 7.2.3 | My virtual world actions tests | 99 |
| 7.3 | General evaluation | 100 |
| V | Conclusion and future work | 103 |
| 8 | Conclusions and Future work | 104 |
| 8.1 | Conclusions | 104 |
| 8.2 | Future Work | 105 |
| VI | Appendix | 107 |
| A | Web site | 108 |
| B | Initialization and mission actions tests | 114 |
| C | My virtual world action tests | 120 |
| D | Web page of the project and Questionnaire | 124 |
| E | UML | 127 |
| | Bibliography | 131 |

Acknowledgement

I would like to express my gratitude to my supervisor, Mr Frank H.R. Fitzek, for giving me the opportunity to work on this project and for his numerous suggestions on how to improve the application. I also want to thank my family and friends for their continuous support through the completion of the project. Special thanks to those who gave me the feedback on the application and thus helped to improve it.

Part I

Introduction

Chapter 1

Spider concepts

1.1 Introduction and Motivation

The development of mobile devices and communication technologies has induced professionals and companies to look for new services for mobile phones. In recent years we have witnessed the discovery of numerous innovative services for mobile phones thanks to the prior creation of new technologies. The invention of these technologies is bound to lead to the further spread of new services in the future.

A huge achievement which indisputably reduced distance between people living in different parts of the world was the creation of the Internet. This invention enabled people to communicate without barriers, and soon on the basis of this technology new services have been created, just to mention such programs as the Messenger, Skype and Face Book. All of the mentioned programs share a common characteristic that they enable people to create a social network life: the users can stay in touch with friends from the whole world, share pictures, talk, chat and send messages, and look for new acquaintances. The current development of numerous information and communication technologies allowed to create similar services also for mobile phones. On Thursday, 16th August 2007 the CBB news informed that a group of university scientists has created a tool which can use the unique ID of Bluetooth devices, like mobile phones, to build new friendship networks. This information suggests that this kind of services will be implemented also on mobile phones. There is a notable difference between services based on the idea of social networks operating in the internet and in the mobiles; in case of internet the space containing information is enormous and therefore it may be difficult to find needed contextual information; in case of mobiles in turn this space is restricted to the range of the Bluetooth device. Moreover, using the new wireless technologies in mobiles allows for data sharing in peer-to-peer networks with communication links created temporarily in an ad hoc manner.

The project named Spider shall explore this technology by developing an application based on a new communication architecture. This architecture should allow devices in cellular controlled networks to establish direct connections between them using the short range communication. The idea behind Spider is to allow customers to exchange data and to establish social interstructures. This report presents the way of utilizing the new technologies to make a service based application that enables users to make spontaneous collaborative networks.

1.2 What is Spider?

Before going to the detailed description of the technical side of the Spider project, it is worth to provide a general overview of the developed application. This will facilitate the full understanding of the program.

The Spider constitutes a program for mobile devices which basically gives an opportunity to its users to exchange their profiles and messages with each other and invite other users to their

social networks. In particular, users can create their personal profiles and register them in a database on a server using an Internet access connection. A part of the information given in the profile (ex. username, age, nationality) is available to all users while the complete information is available only to those who receive an authorization from the respective user. Moreover, using Internet access technologies the users are able to send messages and invitation to other users, and additionally they can add new friends to their list and also do the search in the database on the basis of certain parameters like gender or nationality. An alternative option for users to the one based on Internet access connection is the Bluetooth technology, which allows users to enjoy the same options and functions of the application. Customers can also search other users in their range using the partial information stored on the phone. Additionally, the users will receive a notice when one or more friends will appear in their Bluetooth range.

Furthermore, under the Spider project a web site has been created to provide an alternative possibility to register a user, search people in the database as well as use all the other options that the application created for mobiles provides. Additionally, thanks to the Internet access technology in mobile devices it is possible to download and upload data from and to the server in order to synchronize off line changes. This solution allows to avoid situations, where for instance certain parameters are modified or new friends are added to the list through the web site, and these changes are not updated and visible in the mobile. Thanks to the synchronization between the mobile and the server these problems are avoided.

1.3 Preliminary architecture proposal

This section focuses on the way the Spider application is going to be implemented. The implementation of the Spider concept described in the previous section requires a particular design. The architecture adopted for development of Spider is described below.

The architecture of the software application developed in this thesis is based on the classical Client/Server model. The *Client/Server* separates the application's functionality into two distinct parts. A Server is a kind of command processing engine: it waits for a command from a Client, executes the service that corresponds to the received command, and eventually returns the result to the Client. Usually servers do not have graphical user interfaces [1].

Instead, Clients normally do have user interfaces which accept requests received from a person, verify the correctness of these requests, handle the details of establishing the connection with the Server and dispatches commands to the Server. Next, after receiving an appropriate answer from the Server, the Client elaborates the response and presents it in the form which is understandable to the users.

The *Client/Server* technology is used for several reasons:

- The Client and the Server are two different programs so they operate independently. They are linked only through the message that they exchange
- The Client and Server programs can be implemented separately
- The Client and Server generally function on different computer platforms
- The two programs can be improved independently
- Multiple clients may be connected to the server at the same time

The software architecture is shown in the Figure 1.1. The server in the top of the picture contains a database which includes information regarding all the users of the application. The other elements of the picture represent an assortment of clients. Users of the application can gain access to the Sever through a web site or through mobile devices.

The cooperation between the Client and the Server part looks as follows: a user sends a request through the client to the server. If the Server accepts the connection with this Client, it processes the request by sending a query to the database. Then the database gives back its

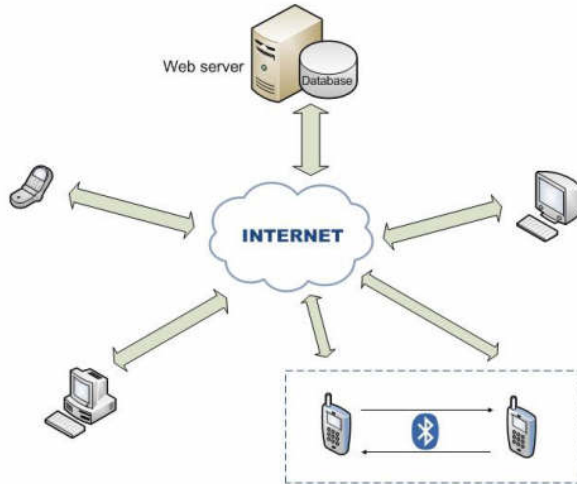


Figure 1.1: Spider architecture

response and the server processes it. Finally, the server is ready to send the response back to the Client. After that the Client presents the response received from the Server in a way which is understandable for users of the application.

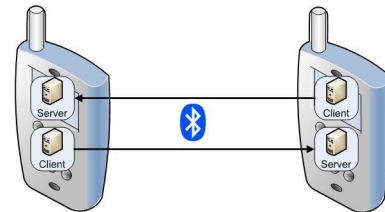


Figure 1.2: Bluetooth communication

Figure 1.2 constitutes an enlargement of the rectangle located in the bottom of the Figure 1.1. As already explained in section 1.2, users can communicate either via Internet access or via Bluetooth. Figure 1.2 refers only to the communication via Bluetooth. In each mobile phone there is a Server and a Client - they are running on the same device. The details of their communication will be given in the successive chapters of the report.

1.4 Thesis overview

This report is divided into five parts: Introduction, Prestudy, Implementation, Testing and Evaluation, and finally Conclusions and Future work. The previous sections of the thesis constituted an introductory overview on the project and provided a short description of its main functions and applications. Additionally, the general architecture proposal to develop the application was presented. The following Prestudy part will focus on analysis of potential resources needed to implement the application and present motives for choosing particular resources for the creation of the program (Chapters 2 and 3). The Implementation part will concentrate upon the server side (Chapter 4) and the client side, which will be divided into the internet connectivity and Bluetooth connectivity (Chapters 5 and 6 respectively). The next part will depict conducted tests of the application as well as the obtained results (Chapter 7). The last part of the report constitutes a summary of conclusions drawn during the development of the thesis as well as discussion on further development of the application (Chapter 8). Additional information regarding deeper exploration of several concepts touched upon in the report will be provided in the Appendix.

Part II
Prestudy

Chapter 2

Technology evaluation

In the section 1.3 the overall Client/Server architecture of the project was presented. To build that architecture five resources are needed:

1. Application Server
2. Database
3. Wireless technologies
4. Target Device
5. Development platform

The current section explains reasons for choosing particular kind of resource from the above given list.

2.1 Application server

The first resource which is needed to construct the Client/Server architecture is an application server. There are numerous types of application servers available, however for this particular project the *Sun Application Server* was chosen. The reasons of this choice are mainly related to its performance. The *Sun Application Server* is really fast and provides a NetBeans integration (the integrated development environment used in the project), which enables rapid development of web applications leveraging multiple languages including Java (the development platform used in the project) [2].

2.2 Database

After choosing the application server the second resource which has to be chosen is a database. The choice of a specific database type for the project was done among *Oracle*, *PostgreSQL* and *MySQL*.

It is not possible to depict one of these three databases as indisputably the best one for all cases. The choice of the most appropriate database for a specific project depends on the actual aim and tasks of the application. Due to the fact that what is needed in this project is a web application, the right choice between *MySQL* and *PostgreSQL* is the first one because it is faster than *PostgreSQL* and it is designed to work well with Web-based servers [3]. Now the choice is restricted to *Oracle* and *MySQL*. The speed was the main reason to discard *PostgreSQL*, but this argument does not apply to *Oracle*, because *Oracle* is also a very fast database. However, the installment of *Oracle* is complex, and additionally it is difficult to tune, while *MySQL* is easily deployable. For these reasons Spider adopted *MySQL* to develop the database constituting a part of the project.

Other advantages of *MySQL* which were also taken into consideration are the following. The first notable advantage of *MySQL* which was already mentioned is the **speed**. In fact *MySQL* executes most of the queries much faster than other database systems. Moreover, it requires little maintenance and administration other than adding or changing user permissions and creating or removing databases. Additionally, *MySQL* is **portable**; it runs in platforms like Unix, Windows and MacOS X. It is also **scalable**, which means that it can run in systems varying in size: from embedded systems to large multiprocessor Unix servers hosting databases with tens of millions of records. Thanks to this scalability it is possible to run a copy of *MySQL* on developer-class machine and later use the same database system on large machine in production. Another important property is that *MySQL* is **flexible**: the user is able to choose the table type needed for his software requirements. All these reasons influenced the choice of the *MySQL* as the most appropriate resource for the project [4].

2.3 Wireless technology

A current mobile device user has typically numerous different wireless technology options to choose from, each of them having strong and weak points. In Table 2.1 a comparison of major competing wireless technologies is shown.

| Wireless Techn. | Advantages | Disadvantages |
|------------------|--|--|
| WIFI | <ol style="list-style-type: none"> 1. Freedom of movement 2. Permanent access to Internet 3. Many reliable WIFI products on the market | <ol style="list-style-type: none"> 1. High power consumption 2. Interference highly impact performance 3. No service discovery |
| IrDA | <ol style="list-style-type: none"> 1. Available in many devices 2. High bandwidth 3. Low power consumption | <ol style="list-style-type: none"> 1. Requires direct sight between devices 2. Very short range (< 1 m) 3. Not network - oriented technology |
| Bluetooth | <ol style="list-style-type: none"> 1. Robust to adverse propagation condition 2. Low energy consumption 3. Many reliable Bluetooth products on the market | <ol style="list-style-type: none"> 1. Short range 2. Long connection time 3. Comply users have it switched off |

Table 2.1: Comparison of wireless technologies

A crucial factor taken into account in choosing the wireless technology for Spider is the level of power consumption, since this application is supposed to run on mobile phones. This argument gives already an important advantage of *Bluetooth* and *IrDA* over *WLAN*. In contrast to *WLAN*, *Bluetooth* is more common in mobile phones and it offers more services and facilities to connect mobile phones. In contrast to *IrDA* in turn, *Bluetooth* offers a bigger communication range and does not require maintaining a direct line of sight. On the basis of analysis of attributes of all wireless technologies it was decided that *Bluetooth* would be the most appropriate communication technology for the Spider application.

2.4 Target Device

The next resource that had to be chosen for this project is the device. For the project two different options were considered: *PDA*s (*Personal Digital Assistant*) and *Mobile Phones*. Even if the *PDA*s are in general superior to *mobile phones*, the use of *mobile phones* is much

more common and therefore this option was chosen for the Spider application.

After defining the mobile phone as the target device, it is important to choose the Operative System (OS) running on the mobile phone.

The most common OS running on mobile phones are:

1. Symbian OS
2. Linux
3. Windows mobile
4. RIM BlackBerry
5. Palm OS
6. OS X - Apple

This section will not describe all these Operative systems, but will focus on the reasons why one of them has been chosen.

A crucial factor in choosing an OS is related to the market of mobile phones. If the best seller is chosen, the application is able to run in the highest per cent of mobile phones being in use.

Figure 2.1 shows that Symbian OS is dominant on the market. In 2007, 73% of the sold mobile phones were equipped with the Symbian OS, which constituted an increase of 17% from 2004.

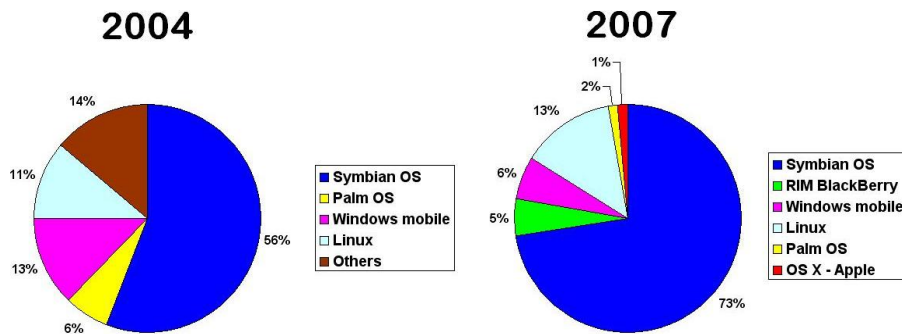


Figure 2.1: Market of the operative systems. "Source: Gartner"

This is a good reason to choose the target device which runs Symbian OS.

Figure 2.2 shows all the companies that adopted Symbian OS as operative system in their devices.

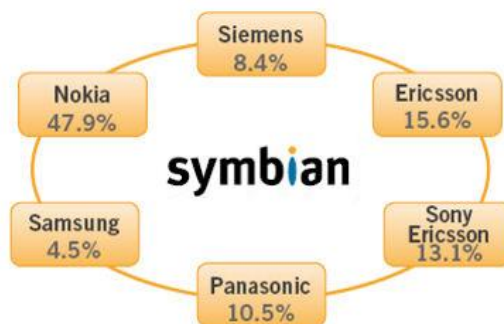


Figure 2.2: Symbian Shareholders [5]

Among companies using Symbian OS as the operative system of their devices, Nokia sells the biggest amount of mobile devices and has a dominant position in the market. For these reasons Spider decided to adopt a Nokia mobile phone as the target device. Specifically, the Nokia N95 was chosen. This device is equipped with Symbian OS S60 (Serie 60 User interface).

2.5 Development platform

The purpose of this section is to study and analyze which mobile programming languages are the most appropriate for the implementation of Spider. In case of Symbian S60, the choice of development platforms is limited to the following three: *Symbian C++*, *Python for S60* and *Java 2 Micro Edition*.

| Development Platform | Advantages | Disadvantages |
|-----------------------|---|---|
| Symbian C++ | <ol style="list-style-type: none"> 1. The only language allowing low level programming 2. One of the most powerful languages for mobile development 3. Memory usage and cleanup 4. Performance: C++ applications run natively on the device, so they are much faster. | <ol style="list-style-type: none"> 1. Complex to program 2. Difficult memory management (manual control) 3. SDKs are not easy to use |
| Python for S60 | <ol style="list-style-type: none"> 1. Script language: more instructions in less space 2. Easy to learn 3. Reach set of APIs | <ol style="list-style-type: none"> 1. Slower execution than Java ME 2. Used in few mobile phones (only S60) 3. Limited access to the resources |
| Java2ME | <ol style="list-style-type: none"> 1. Easy access to learning materials 2. Used in most mobile phones 3. Portable: write once run anywhere | <ol style="list-style-type: none"> 1. J2ME is limited in comparison to J2EE 2. Slower execution than Symbian C++ |

Table 2.2: Comparison of development platforms

Based on the comparison of both advantages and disadvantages of all three languages the best option was chosen for Spider application. *Python* was assessed as a language which is not suitable for the application. The main reason to discard *Python* is the fact that it is not common in mobile devices, and the spread value constitutes one of the most important factors by choosing resources for Spider project.

Table 2.2 shows that *Java 2 Micro Edition* and *Symbian C++* offer the most crucial qualities and advantages. The key reason to choose *Java Micro Edition* is the fact that *Symbian C++* runs only on mobile phones equipped with Symbian OS while *Java ME* runs on majority of mobile phones using also other operative systems.

Therefore, *Java ME* was preferred to *Symbian C++* and *Python* programming languages in development of the Spider application.

Chapter 3

In depth technology study

The previous chapter presented reasons for choosing particular type of resources to build the proposed architecture of the Spider project. Once all the resources have been chosen, the following section focuses on giving a theoretical overview and presenting the technical details of the resources.

3.1 Sun Java Application Server

Java 2 Platform, Enterprise Edition (J2EE) provides the foundation for building reliable, scalable, and manageable applications. *Sun Java Application Servers* are based on the J2EE and constitute a compatible platform for developing and delivering server side Java applications and web services [2].

The *Sun Java System Application Servers* are available in different editions, each designed to provide specific functionality for various usage scenarios and service levels. This particular project is based on the use of the *Sun Application Server 9.1*. This edition of *Sun Java System Application Server* includes a highly scalable HTTP connection handler, and it can handle thousands of connections with a small number of threads [6].

3.1.1 Sun Java Application Server architecture

Application Servers use a multi-tier distributed model, which consists of three major parts: the Client Tier, the Middle Tier and the EIS Tier (Figure 3.1).

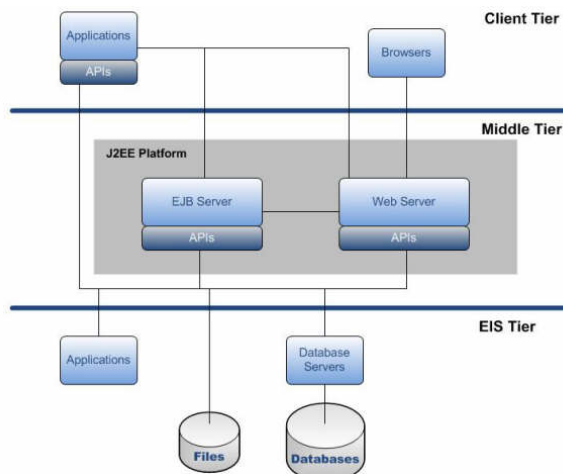


Figure 3.1: Application Server Model

The **Client Tier** is composed of one or more applications or browsers. Clients can access web applications by communicating with the Web server via various protocols, for instance via

HTTP. For each protocol the Application Server offers separate listeners. Each listener has an exclusive use of a specific port number [7].

The **Middle Tier** consists of a Web Server and an Application (EJB) Server. The behavior of these components is very similar: they both respond to requests received from clients. The only difference between the two components is that the Web Server hosts pre-made HTML documents which may be requested by web browser clients, while the Application Server - after receiving a request - interprets the request and starts a java servlet to do specific computation, and then returns the result as HTML document - just as a "normal" web server. In other words, the Application Server uses java servlets to produce HTML and a simple Web Server uses static (pre-made) HTML documents or files [7].

The **EIS Tier** is the Enterprise Information System (EIS) which contains the existing applications, files and databases [7].

3.1.2 Sun Java Application Server Security

This chapter describes some core security issues related to the application server. Security regarding the application server could be divided in two to areas:

1. Protecting data
2. Secure Client/Server communication

As far as protecting data is concerned, this security area regards ways to prevent unauthorized access or damage to data in storage or transit. The Sun application server has certain built in security features, which include cryptography, authentication and authorization, and public key infrastructure [8].

Among these, the central concepts of application server security are authentication and authorization. Authentication is a measure that allows to verify if an entity is who it claims to be using security credentials, which usually take a form of a username and password. Thanks to this mechanism the application server is able to determine whether an entity can be granted access to a protected resource. Once an entity gets the authentication, it can perform different actions depending on the authorization mechanism. One can understand the authorization concept as a set of different authorization levels, each of them allowing for accessing specific resources. Some entities may have access to all resources while the access of others may be limited.

As far as the communication between clients and servers is concerned, a secure communication is recommended whenever sensitive data are exchanged. A security of communication can be provided by using the Secure Sockets Layer communications protocol (SSL).

SSL communication protocol is based on three fundamental points:

- SSL server authentication
- SSL client authentication
- Encryption of SSL connection

SSL server authentication allows a user to confirm a server's identity, while SSL client authentication allows a server to confirm a user's identity. An encrypted SSL connection provides a high degree of confidentiality since it requires all information exchanged between a client and a server to be encrypted by the sending software and decrypted by the receiving software. Confidentiality is crucial for both parties in any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism detecting any manipulation of data during their transit [9].

3.2 MySQL

MySql is an open source, multithreaded, relational database management system. Since 2000 it has been available to general public under the General Public License (GPL) and therefore it became one of the most popular open source SQL databases. According to the statistics of the company that developed *MySql* (MySQL AB), there are more than 4 million people who use *MySql* and there is an average of 35,000 downloads per day of *MySQL* installation software from the official and mirror site of *MySQL* [10]. Besides the fact that *MySQL* is gratuitous and widespread among the public, it has other relevant features [11]:

1. *MySQL* is a **relational database management** system which stores data in separate tables, in this way adding speed and flexibility.
2. *MySQL* software is an **Open Source**. It gives the opportunity to use and modify the software. Everybody can download the software from the official and mirror site for free
3. *MySQL* is also **easy to access** from other languages like Java, C, C++, PHP, using specific libraries and APIs for connecting to MySQL

3.2.1 Mysql Architecture

MySQL like all the rest of database systems refers to the general Relational database management system (RDBMS) architecture. This architecture is composed of three main components as it is shown in Figure 3.2:

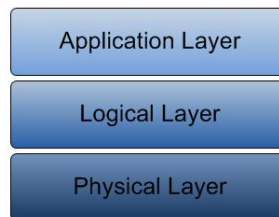


Figure 3.2: MySQL Architecture

The **Application Layer** represents the interface for all the users of the system. Through this layer users can interact with the database server.

The **Logical Layer** represents the core of the RDBMS. Figure 3.3 shows *MySQL Logical layer*.

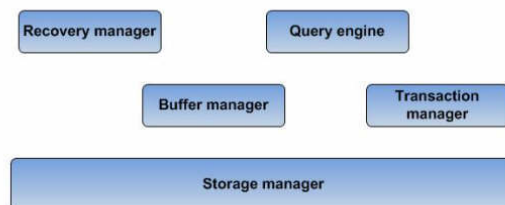


Figure 3.3: The Logical Layer

As is shown in Figure 3.3, this layer is divided into five different modules: the Query Engine, the Recovery Manager, the Transaction Manager, the Storage Manager and the Buffer Manager. These components will be described separately [12]:

- The Query Engine is a process which decomposes the SQL commands it receives, checks the correctness of the syntax, finds the index which should be used to retrieve the data in a quick and efficient way and creates the requests for the other components to retrieve the records on the base of the received information. It is possible that the user types the same query more than one time. If the data in the database has not changed the result record is the same. *MySQL* uses an efficient caching mechanism called Query Cache. The data from a query are placed in a cache, and when a similar query is issued, these data are returned.
- The Recovery Manager performs really important tasks which is keeping and coping data in case of a loss of data.
- The Transaction Manager is in charge of facilitating concurrency in data access. Its task is to ensure that multiple users can access the data without corrupting or damaging them. It uses a Lock Manager subcomponent to place and release the locks during transactions.
- The Storage Manager interfaces with the operating system to write data to the disk in an efficient manner.
- The Buffer Manager deals with the memory management issues. The Buffer manager contains the cache, and hence new records can be cached. Thanks to this possibility, the needed data are requested from the Storage Manager and placed in the buffer before being sent to the Query engine.

The last layer is the **physical layer** which is in charge of storing the information. Data are kept in a secondary storage and they are accessible via the storage manager. Possible kinds of data that can be stored are as follows [13]:

- Data files, which store in the database data regarding the users
- Data dictionary, which stores metadata regarding the database structure
- Indices, which provide fast access to some data items
- Statistical Data, which contain information regarding the data in the database. These statistical data are used by the query processor to find the most efficient way to execute a query
- Log Information, which is used to keep track of executed queries. In case of a system crash, the recovery manager can use this information to recover the database.

3.2.2 MySQL Security

Users can access the *MySQL* database in two ways: by connecting to the *MySQL* server or by connecting to individual objects like tables. *MySQL* manages user authentication through user tables, which check whether a user has logged on with the correct username and password, and whether the connection is originating from an authorized TCP/IP address [12].

Some data such as password are really important and they should be transmitted in a secure way. But being transmitted by the network they are prone to interception by others. For this project the newer version of *MySQL* was adopted since it provides greater security than the old versions. However, in the new version of *MySQL* such data as query results are still not encrypted. If the developer wants to increase the security of these data, he can use the Secure Socket Layer encryption protocol to make the data transmission safer over the internet and other public network infrastructures [12].

3.3 Bluetooth

Bluetooth is an innovative technology which provides a way to connect and exchange information between devices over a secure, globally unlicensed short-range radio frequency. It

is considered a wireless PAN (Personal Area Network) technology that offers fast and reliable transmission for both voice and data between different types of devices (PDA, headphones, phones, printers, etc).

When *Bluetooth* was implemented, its goal was to replace cable in small device with communication capabilities. In fact the three main concepts of *Bluetooth* are: small size, minimal power consumption and low price. Moreover, this technology enables an easy file sharing between devices equipped with *Bluetooth*. It also provides automatic wireless synchronization with other *Bluetooth* enabled devices. Furthermore, it is possible to use the internet connectivity when a Bluetooth device is connected to another one with internet access capability.

This section explores the major features of *Bluetooth* technology. First, the *Bluetooth* protocol stack is presented and then different kinds of *Bluetooth* connections are examined. Finally, the *Bluetooth* security is described.

3.3.1 Bluetooth Protocol Stack

To better understand the way the *Bluetooth* technology operates, it is worth to have a brief look on its internal module and their relevant tasks and functionalities.

Bluetooth protocol stack is composed of two layers:

1. Lower level layer
2. Upper level layer

A crucial element of the lower level layer is the Bluetooth Radio, which is in charge of the modulation and demodulation of data into RF signals for transmission in the air. The Radio Layer defines the physical characteristics of the *Bluetooth* device. The range of *Bluetooth* depends on the power class of the radio. Table 3.1 shows possible classes which define the *Bluetooth* range [14].

| Device | Range |
|---------|-----------------------------|
| Class 1 | Maximum range of 100 meters |
| Class 2 | Maximum range of 10 meters |
| Class 3 | Maximum range of 1 meter |

Table 3.1: Bluetooth Classes

Obviously a more powerful radio implies larger power consumption. Due to the fact that the issue of power consumption is crucial for mobile phones, Class 1 is not recommended for this kind of devices, although it still remains a possible option.

The simplified architecture of the upper level layer is presented in Figure 3.4.

The Host Controller Interface (HCI) is the interface between the lower and the upper level of the protocol stack: everything below this interface is implemented in hardware and everything above in software. Some protocols among the ones presented in Figure 3.4 are specific to *Bluetooth*, for example Logical Link Control and Adaptation (L2CAP), while others are adopted, for instance OBEX, PPP, IP, UDP and TCP. These adopted communication protocols are not going to be described, because they are not used in the project.

L2CAP handles all data transmission from the upper layers. It is in charge of two important tasks: first, segmenting data into packets for transmission, and second, re-assembling data packets when they are received.

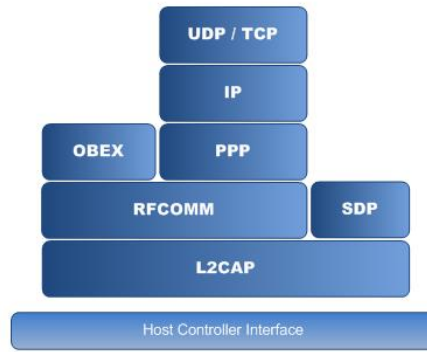


Figure 3.4: Bluetooth Protocol Stack - Upper level

Radio Frequency Communication (RFCOMM) is a simple set of transport protocols which is made on top of the L2CAP. It is a cable replacement protocol and it simulates the functionality of a standard serial communication port.

The Service Discovery Protocol (SDP) is responsible for discovering Bluetooth services offered by remote devices [15].

3.3.2 Bluetooth Connection

In order to connect with *Bluetooth* devices there are several steps that have to be followed. First of all an inquiry should start. As the response to the inquiry all devices which support a *Bluetooth* technology will be shown. Then a service discovery starts to investigate if the discovered *Bluetooth* devices are equipped with the desired service. Once that these requirements have been satisfied, the security setting has to allow for this connection. If all of these conditions are met the connection can be established between devices. Once the connection has been established different scenarios are possible (Figure 3.5).

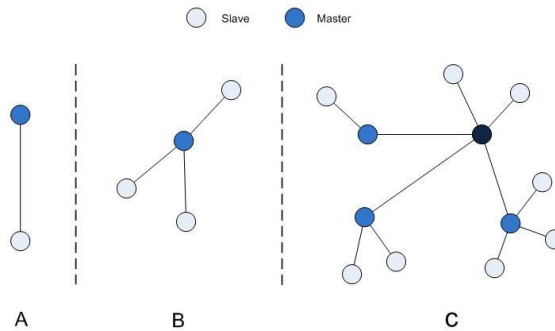


Figure 3.5: Bluetooth piconet and scatternet scenarios

Figure 3.5-a shows a Point-to-point connection between two devices. The device which started the connection is called master while the other device is called slave. Master/Slave is a model for communication protocol in which one entity (master) controls other entities (slave). Once a master and a slave establish a relationship, the master is always in charge of the control. In *Bluetooth* connections the network is established just for the current task and destroyed after the data has been transferred.

A master can be connected with one or more slaves. This is the case of Figure 3.5-b where the scenario is Point-to-multipoint connection between a master and three slaves. In this case the data rate is limited because it is not possible to have a full data rate for all the links. Figure

3.5-a and Figure 3.5-b show a *Bluetooth* network which is called piconet.

One device can be connected with one or more piconets at the same time: this scenario is called scatternet and it is shown in Figure 3.5-c. A scatternet is formed either when there are too many devices that wish to connect, or when some devices are out of range of the master, but within range of one of the slaves. There is only one master within a piconet, and all other devices are slaves. This is still true for scatternet scenarios, but if a device participates in two piconets it can be both a slave and a master. It then plays the role of a slave in one piconet, and the master in another piconet. This is what happens when a scatternet is formed: one of the slaves in the first piconet is also assigned the role as the master of the second piconet, this device is then the link between the two piconets [1].

3.3.3 Bluetooth Security

One of the most crucial features of *Bluetooth* is the security. *Bluetooth* offers the following forms of security [15]:

- Authentication
- Authorization
- Encryption

Authentication ensures the identity of *Bluetooth* devices. It is based on a PIN number shared between devices. Once that 2 devices enter an identical passkey in both devices, these two devices create a trusted relationship called **pairing**. As long as both devices are paired, the PIN code will be not necessary anymore for future connections.

Authorization determines if a device is enabled to have access to a specific service. This is always required to the users. On current Symbian OS, devices allow static authorization by the remote device which has been authenticated. It means that for trusted devices the access to service is required automatically while untrusted devices need an authorization procedure.

Encryption protects sensitive data from eavesdropping. The length of the encryption key can be between 8 and 128 bits.

These three forms of security can be used in various combinations. The Authorization and Encryption can be used only when the Authentication is applied.

Thus there are four possible combinations of security forms:

1. Authentication
2. Authentication and Authorization
3. Authentication and Encryption
4. Authentication and Authorization and Encryption

3.4 Java 2 Micro Edition

Java 2 Micro Edition (J2ME) is the newest and smallest addition to the Java family. The other members of the Java family are the Java 2 Standard Edition (J2SE) and the Java 2 Enterprise Edition (J2EE). The former is intended for conventional desktop applications development, while the latter one is specifically intended for building distributed applications with emphasis on the server side development and web applications. *Java 2 Micro Edition* is intended to build applications running on mobiles and other embedded devices. [16]. Figure 3.6 presents the appropriate Java editions for different kind of devices.

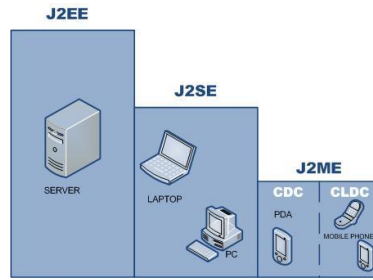


Figure 3.6: Java family

In the next section an overview of the *Java 2 Micro Edition* architecture is going to be given, taking into consideration the particular configuration used in this project. Next the MIDlet, which is the MIDP application, will be examined.

3.4.1 Java Architecture

Figure 3.7 shows the architecture of Java. Here it is visible that *Java 2 Micro Edition* is composed of a set of configurations, profiles and optional packages.

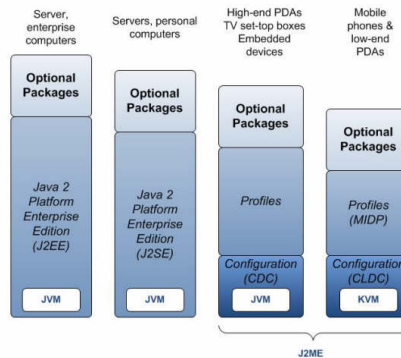


Figure 3.7: Java architecture

In *Java 2 Micro Edition* two different configurations exist:

- Connected Limited Device Configuration (CLDC)
- Connected Device Configuration (CDC)

The first one is intended for more limited devices like mobile phones, while the second one is intended for devices with more memory and faster processors. Figure 3.8 shows the relationship between CDC, CLDC and J2SE: all the the main functionalities of CDC and CLDC are inherited from J2SE and moreover, all the features implemented in the CLDC are implemented in the CDC as well, in order to obtain upward compatibility between the two J2ME configurations [16].

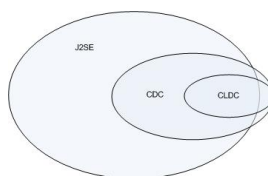


Figure 3.8: J2ME configurations

Due to the fact that in this project the CLDC is used, only this configuration is going to be presented. Figure 3.9 shows the layer model of the CLDC configuration.



Figure 3.9: CLDC Configuration Architecture

Kilobyte Virtual Machine (KVM) is placed at the bottom of the architecture in Figure 3.7. It is implemented in C and it is responsible for interpretation of the Java byte-code and translating this into native system calls. The KVM is a highly portable Java Virtual Machine designed for small devices as cellular phones. It is able to run with only a few hundred kilobytes of memory [17].

CLDC Library is the layer above the KVM. CLDC consists of the Java packages presented in Table 3.2.

| Package | Short description |
|-----------------------------------|---|
| <code>java.io</code> | Provides for system input and output through data streams |
| <code>java.lang</code> | Provides classes that are fundamental to the design of the Java programming language |
| <code>java.lang.reflect</code> | Provides support for weak references |
| <code>java.util</code> | Contains the collections framework, legacy collection classes, date and time facilities and miscellaneous utility classes |
| <code>java.microedition.io</code> | The classes for the generic connections |

Table 3.2: The Java packages in CLDC [18]

Only two CLDC versions exist: CLDC 1.0 and CLDC 1.1. CLDC Version 1.0 (JSR 30) is the first release of the CLDC specification, which provides a compact virtual machine and basic libraries for resource-constrained devices. CLDC Version 1.1 (JSR 139) is a backward-compatible revision of the CLDC 1.0 specification, with the same target, small devices with limited resources, and the same key objective, maintaining a tight footprint. Its many additions and enhancements include support for floating-point math and weak references [19].

Above the CDLC Library the **Mobile Information Device Profile (MIDP)** library layer is placed. This is a set of APIs for a specific class of devices. Currently there are two versions of MIDPs: MIDP 1.0 and MIDP 2.0, the latter one being a revised version of the former one.

MIDP 2.0 includes new features such as an enhanced user interface, multimedia and game functionality, greater connectivity, over-the-air (OTA) provisioning, and end-to-end security. MIDP 2.0 is backward compatible with MIDP 1.0, and continues to target mobile information devices such as mobile phones and PDAs [20].

Table 3.3 shows the collection of APIs of MIDP 2.0. In case the particular library is also included in MIDP 1.0 version, this is depicted in the Table 3.3.

| Package | Short description |
|---|--|
| java.microedition.io | Classes for the Generic Connection framework |
| javax.microedition.lcdui | The UI API provides a set of features for implementation of user interfaces for MIDP applications |
| javax.microedition.lcdui.game | The Game API package provides a series of classes that enable the development of rich gaming content for wireless devices. This package is not included in MIDP 1.0 |
| java.util | Contains the collection framework, legacy collection classes, date and time facilities and miscellaneous utility classes |
| javax.microedition.media | The media library includes features such as basic tone generation and WAV file playback. This package is not included in MIDP 1.0 |
| javax.microedition.media.control | Provides simple media control such as volume control. This package is not included in MIDP 1.0 |
| javax.microedition.midlet | The MIDlet package defines Mobile Information Device Profile applications and the interactions between the application and the environment in which the application runs |
| javax.microedition.pki | Certificates are used to authenticate information for secure Connections. This Package is not included in MIDP 1.0 |
| javax.microedition.rms | Provides a mechanism for MIDlets to persistently store data and later retrieve it |

Table 3.3: MIDP 2.0 [21]

Above the profile layer the optional packages are placed. An optional package is a set of technology-specific APIs that extends the functionality of a Java application environment. CLDC supports a number of optional packages that allow product designers to balance the functionality needs of a design against its resource constraints [19].

Table 3.4 shows the possible optional packages which can give the device the possibility to support additional technologies.

| Package | Short description |
|------------------|--|
| JSR - 82 | Java APIs for Bluetooth: Provides API's through which the application programmer can access Bluetooth |
| JSR - 211 | Content Handler API: Let an application invoke registered J2ME and non-Java applications by URL, by content type, or by content handler ID |
| JSR - 135 | Mobile Media API: Mobile Media API (JSR-135) Specification, defines the Multimedia API for the Java TM 2 Platform, Micro Edition (J2METM) |
| JSR - 172 | Web Services Specification: Provide an standard access to J2ME web services |
| JSR - 177 | Security and Trust Services APIs: Provide J2ME applications with APIs for security and trust services through the integration of a Security Element |
| JSR - 209 | Advanced Graphics and User Interface: Provides an advanced graphics and user interface |
| JSR - 66 | RMI: Provides Java TM platform to Java platform remote method invocation for Java devices and interoperates with J2SE RMI |
| JSR - 120 | Wireless Messaging API Provides standard access to wireless communication resources, designed to run on J2ME configurations and to enhance J2ME profiles with unique functionality |
| JSR - 75 | PDA: Produces two separate optional packages for features commonly found on PDAs and other J2ME mobile devices: one for accessing PIM data and one for accessing file systems |

Table 3.4: Optional packages [18]

3.4.2 MIDlet

MIDlet is the MIDP application. *MIDlet* differs from a normal java application in one particular point: it replays the `main()` method of the J2SE application by the `startApp()`,

pauseApp() and destroyApp() methods which respectively start, pause and destroy the application. A life cycle scenario is created, which is controlled by the Application Manager System (AMS). The AMS is part of the device operating system and it moves the application through the different states which are shown in Figure 3.10.

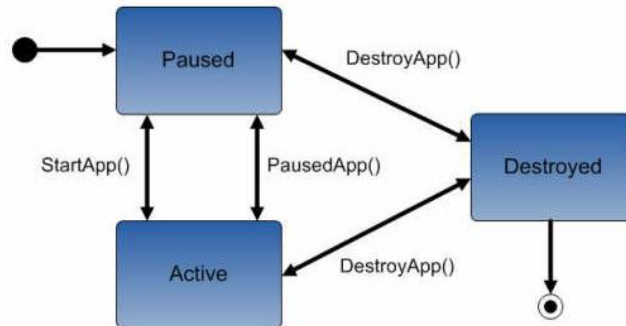


Figure 3.10: MIDlet lyfe-cicle

The *MIDlet* can be placed in three different states:

1. Paused: The *MIDlet* is in a sleeping state. *MIDlet* can enter in this state in the following cases:
 - When the *MIDlet* is initialized
 - When it is in the active state and the pauseApp() method is called by the AMS
 - When it is in the active state and the notifyPaused() method has been called and successfully returned
2. Active: the *MIDlet* is running and functioning normally.
 - This state is entered after the startApp() method has been invoked.
3. Destroyed: the *MIDlet* has terminated and released all the resources. *MIDlet* enter in this state in the following cases:
 - destroyApp() method has been called by the AMS and successfully returned.
 - when the notifyDestroyed() method successfully returns.

3.5 Nokia N95

Due to the fact that mobile phone is the target device of Spider project, this section provides the characteristics of the model of the chosen phone - *Nokia N95*.



Figure 3.11: Nokia N95

Operating system

- Symbian OS 9.2 - S60 3.1

Processor and Memory

- 332Mhz ARM11 processor
- 160 MB Internal memory
- Slot microSD for external memory till 2 GB

Connectivity

- Bluetooth v2.0 + EDR
- Wi-Fi IEEE802.11b/g
- USB 2.0 through Mini USB
- IrDA

Java Application

- Java MIDP 2.0, CLDC 1.1 (Connected Limited Device Configuration (J2ME))

Part III

Implementation

Chapter 4

Server side: Internet connectivity

In this section the server side will be examined. A crucial part of the server is the database which is in charge of storing all the data of users.

Users connect to the database using a servlet. The servlet sends the data to the database and receives appropriate answers which are next sent back to the users.

This chapter first focuses on the database design of the project. This is followed by an in-dept discussion of the servlet and the way users connect to the database.

4.1 Database Design

This section presents the logical scheme of the database and the strategy for its implementation. This is supported by several examples which show the advantages of this strategy for the developer.

4.1.1 Logical schema

Figure 4.1 shows the logical schema of the database.

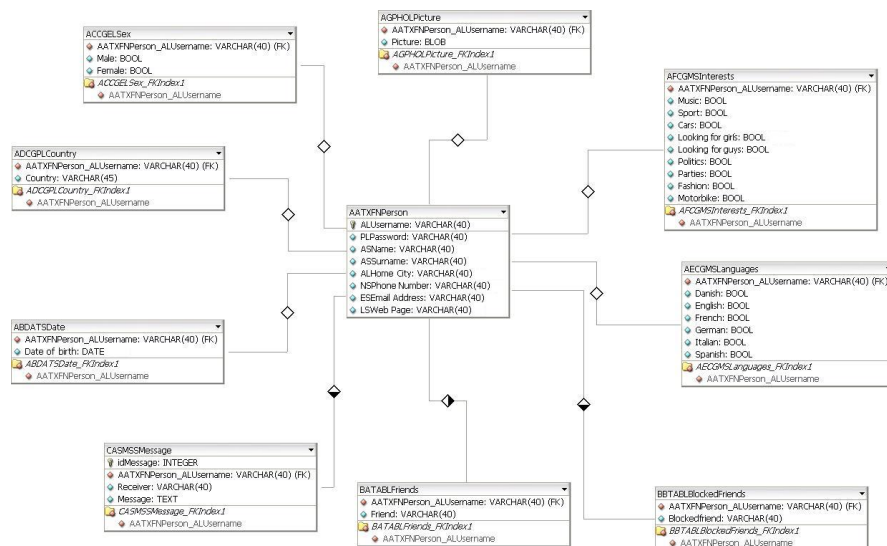


Figure 4.1: Database logical scheme

In the logical schema a box is denoted as table and the primary key is denoted with a key symbol. All the foreign keys are marked (FK). What remain are all instances and after every

instances is possible to see its associate data type (ex.: VARCHAR(40), INTEGER, BLOB, etc).

The relations are described through rhombus. If the rhombus is blank it denotes a one to one relation, otherwise if the rhombus is half filled, it denotes a one to many relation. The one to one relationship indicates that one record of the entity A is associated with one record of the entity B. The one to many relationship indicates the one record of the entity A is associated with one or more instance/s of the entity B.

This database is composed of 10 tables. The first table which will be examined is *AATXFNPerson*. This entity is a part of the whole profile of the user, and contains the following instances which are stored as VARCHAR(40) type:

- ALUsername, which is the username that the user has chosen. This is also the primary key
- PLPassword, which is the password that the user has chosen to log to the application
- ASName, which represents the name of the user
- ASSurname, which represents the surname of the user
- ALHome City, which is the home city of the user
- NSPhone Number, which is the mobile phone number of the user
- ESEmail Address, which represents the email address of the user
- LSWeb Site, which is the home page of the user

This table is connected with a one to one relation with the following tables:

- ABDATSDate
- ACCGELSex
- ADCGPLCountry
- AECGMSLanguages
- AFCGMSInterests
- AGPHOLPicture

All these tables together with *AATXFNPerson* represent the profile of the user. In each of these tables the *AATXFNPerson_ALUsername* is present, this instance is the foreign key which identifies the user in each table. All these entities are connected through with a one to one relationship with the table *AATXFNPerson* are going to be examined.

The first table is *ABDATSDate* which contains the instance *Date of birth* that indicates the date of birth of the user. To store this instance a DATA type is used.

Then the entity *ACCGELSex* represents the gender of the user. The gender could obviously be Male and Female. To store these values BOOLEAN types are used.

The table *ADCGPLCountry* contains the instance Country which represents the home country of the user. A VARCHAR(40) type is used to store this value.

The table *AECGMSLanguages* contains a list of languages that the user can speak. The list of the languages includes : Danish, English, French, Italian and Spanish. To store these values BOOLEAN types are used.

The entity *AFCGMSInterests* represents the interests that the user may have. The list of interests is composed of: Music, Sport, Cars, Looking for girls, Looking for guys, Politics, Parties, Fashion and Motorbike. Also in this case BOOLEAN types are used to store these values.

The table *AGPHOLPicture* contains the instance Picture that is used to store the picture of the user. To store this value a BLOB type is used.

The *AATXFNPerson* is connected with one to many relation with the following tables:

- *BATABLFriends*
- *BBTABLBlockedFriends*
- *CASMSMessage*

The *BATABLFriends* table represents the friend status of the users. If two users are friends they are inserted into this table. The table contains the foreign key *AATXFNPerson_ALUsername* and the instance Friend. These instances identify the relationship of friendship. The Friend instance is set as VARCHAR(40) type.

The *BBTABLBlockedFriends* table represents the invitation status. If some user sends an invitation to another user, they are inserted into this table. The *BBTABLBlockedFriends* table contains the foreign key *AATXFNPerson_ALUsername* and the BlockedFriend instance. The latter one represents the sender of the invitation and the former one represents the receiver of the invitation. BlockedFriend is set as VARCHAR(40) type.

The entity *CASMSLMessage* is used to store messages that the users exchange. It includes the instances: the primary *idMessage* which identifies the message and indicates it as an INTEGER, the foreign key *AATXFNPerson_ALUsername* which is the user who wrote the message, Receiver which is the user who received the message and it is indicated as a VARCHAR(40) type, and the Message which is the text message indicates as TEXT type.

In order to interrogate the database *Query statements* are used. In this project Insert, Delete, Update and Select queries are used. The Insert query is used to insert a new row in the table. The Update query is used to modify a row in the database, while the Delete query on the database is used to delete a row from the database. A Select query is used to interrogate the database.

Sections 4.2 and 4.3 of this chapter more detail are going to be given. It will be explained how the server accesses to the database and answers to client's requests.

4.1.2 Strategy of the database implementation

This section presents the way in which the general concept of the database was implemented.

In case the developer of the application wants to add some data to the profile of the user, he doesn't have to modify any code in the interface of the mobile phone. In fact it is enough to add it to the database instances on the existing tables or create new tables following the requirements of the implementation which are described below.

Basically, the entities of the database are divided in three main categories. These categories are identified by the first character of each table:

1. A: which is used for the whole profile of the user
2. B: which is used to manage the friends status of each user
3. C: which is used to manage the actions of each user

These different kinds of categories are explained below.

Figure 4.3 shows the instances of the table *AATXFNPerson*. For each instance present in this table, if the character (1,2) is L then the value is part of the local profile and has to be stored on the memory of the mobile phone; otherwise it is only part of the full profile and has to be stored only on the database. In the database the *ALUsername*, *PLPassword* and *ALHome City* are part of the local profile and they will be stored in the database and on the memory of the mobile phone. *ASName*, *ASSurname*, *NSPhone Number* and *LSWeb Page* are only part of the full profile and they are going to be stored on the database.

In this project the local profile is called *Business Card*. Values shown on the business card are:

- Username
- Password
- Home City
- Sex
- Country
- Picture

These are mandatory values. The user must insert these values, otherwise he cannot be registered in the application.

Revisiting Figure 4.2, the sequence of characters (2,5) indicates the kind of component to be used in the user interface. There are 5 different options:

1. TXF: means Text Field and indicates that in the user interface a component which has the characteristic of editable text has to be inserted. In this case, the implementation will also examine the character (0,1) of each instance of the table. There are gives 4 different options:
 - A: means Any and includes *ALUsername*, *ASName*, *ASSurname* and *ALHome City*. It means that the user is allowed to enter any text
 - P: means Password and includes only *PLPassword*. It indicates that the entered text is confidential data that should be obscured whenever possible
 - N: means Phone Number, includes only *NSPhone Number* and indicates that the user is allowed to enter a phone number
 - L: means Emailaddr includes only the instance *LSEmail Address* and indicates that the user is allowed to enter an e-mail address
2. CGE: means Choice Group Exclusive and indicates a group of selectable elements which has to be chosen in an exclusive way. It means that only one element can be chosen at one time. In this category only the *ACCGESex* is included.
3. CGP: means Choice Group Popup and has the same features of the CGE choice but with different interface. While all the element of CGE are always displayed, on CGP only the selected element is displayed. Only the entity *ADCGPLCountry* is contained in this category.
4. CGM: means Choice Group Multiple and indicates a group of selectable elements which can have arbitrary number of elements selected at one time. In this category the entities *AECGMSLanguages* and *AFCGMSInterests* are included.
5. PHO: means Picture and allows the application to take a picture of the user. Only the entity *AGPHOLPicture* is included in this category.

The last sequences of characters of each name of the table (6,n) are the name that will be shown in the user interface on the mobile phone.

Category B

After having explained the main category A, which represent the profile of the user, the next step is to explain the main category B, which represents the friends status of each user with other users. In the main category B, two tables are included: *BATABLFriends* and *BBTABLBlockedFriends*.

The way how users create new friends will be examined in details in the client side. Now it is enough to know that users are able to send invitations to each other. Once a user A sends and invitation to the user B they both reside to the *BBTABLBlockedFriends* table till the user B accept or refuse the invitation. If he accepts the invitation they will be inserted into the table *BATABLFriends* and deleted from the table *BBTABLBlockedFriends*. In the case the user B refuses the invitation they will be only deleted from the table *BBTABLBlockedFriends*.

Category C

The last main category is the category C which represents the actions of the user. In this category only the *CASMSSMessage* table is included.

Thanks to this table users are able to send and retrieve messages.

4.1.3 Examples of using the strategy of the database implementation

In this section some examples of using the implementation strategy are given.

Example 1: Add new language

Let us assume that the developer would like to add a new language in the list of language, for instance the Polish language. He has to add to the database the Polish instance in the table *AECGMSLanguages* and when a client connectes, the user interface will be refreshed automatically.

Example 2: Add Zodiac sign

Let us assume that a new editable text would be inserted such as Zodiac sign. The developer has to insert this instance in the *AATXFNPerson* with the following requirements:

1. Indicate the type of text that the user should insert
2. Specify if this parameter has to be a part of the business card or not

Let us assume that the user can insert any type of text and this parameter has to be a part also of the business card. The developer has to insert in the *AATXFNPerson* table the instance ALZodiac Sign.

Example 3: Add photo album

The last example is the case when the developer would like to insert a new category like photo album. The developer has to create a new table with the following requirements:

1. The main category
2. The priority
3. The interface type
4. Specify if this category has to be a part of the business card or not

Let us assume that he would like that this category is not contained in the business card and he would like to show it as the last element. The result will be the table called: *AHPHOSBook Pictures*.

In this way the modifications are really simple and thanks to this design implementation the developer doesn't have to modify the code in the mobile phone, but it is enough to modify the database which reside in the server part.

4.2 JDBC

This section introduces the way the database receives commands to executes queries on the database. To communicate with the database the *Java DataBase Connectivity* (JDBC) is used.

JDBC is a Java API which serves to execute SQL statements. It consists of a set of classes and interfaces written in Java programming language. Thanks to the JDBC, it is possible to send SQL statement to any relational database.

JDBC supports two different models for database access:

1. Two-tier model
2. Three-tier model

In the two tier model the application communicates directly with the database, while in the three-tier model between the application and the database there is an intermediary tier.

In this project a three tier model for database access is used. This model was chosen for Spider application since the use of the middle tier is advantageous due to two reasons. First, the middle tier mantains control over access to data and second, it controls the sorts of updates that can be made to the data. Additionally, the middle tier enables the user to employ a higher-level API, which is then translated by the middle tier into the appropriate low-level calls [22].

Figure 4.4 shows the three-tier model architecture.

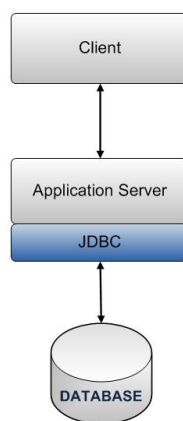


Figure 4.4: JDBC Architecture

In this model the process in which commands are sent to the database is as follows: the client sends commands to the middle tier which then sends SQL statements to database. Next, the database processes the SQL statements and sends the corresponding results back to the middle tier. Finally, the middle tier sends the received results to the client [22].

Server/Database connection

Whenever a server receives a request from a client to execute a query on the database, it needs to perform the following actions:

- Load the JDBC driver class and establish the connection with the database (connection pooling)
- Send a request to the database
- Receive a response from the database

Example code 1 shows the way the server connects to the database.

Example code 1 JDBC connection

```
1: Connection con = null
2: Statement sta = null
3: Class.forName("org.gjt.mm.mysql.Driver")
4: con = DriverManager.getConnection("jdbc:mysql://localhost:port number/name of the db", "root",
   "password")
5: Statement sta = con.createStatement()
6: ResultSet rs = sta.executeQuery("Query")
```

In the line 3, the *Class.forName* method is used to load the JDBC driver class. In the next line thanks to the *API JDBC* a connection with the database is opened. In line 5 the *Statement* object is obtained. This object is used for executing a static SQL statements. In the last line the query is executed and the *ResultSet* object is obtained. A *ResultSet* object is a table of data which includes results obtained from the database.

These are the main actions that the server needs to perform in order to communicate with the database. Once it gets the response of the database, the server has to elaborate them before sending them to the client. In the next section these actions are going to be described in details.

4.3 Servlet

"A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers" [23].

At first this section explores the way the clients create requests. This description is given since in the next step it will facilitate understanding understanding the way the servlet manages the requests and responses to the clients.

4.3.1 Client requests

This section describes the way the clients open connections with the server. When a client tries to connect with the server, it uses an HTTP connection. HTTP (Hyper Text Transport Protocol) is a request/response protocol between a client and a server, where the request parameters have to be set before the connection is established. Clients using the user interface of the mobile phones to create some requests which will be sent to the database. They do it by attaching parameters to the url. URL (Universal Resource Locator) is an address which can identify the position of an object on the web.

The syntax of URL is composed of:

typeprotocol://namehost/namefile

The first part is the kind of protocol. It could be a *http*, *ftp*, *telenet*, etc. The second part of the syntax is the name of the host where the file is placed and the last part is the name and

position of the sought file.

An URL parameter is composed by the following syntax:

$$name = value$$

The first parameter begins with the symbol `?`. If there is more than one parameter, all the other parameters begin with the symbol `&`.

Let us have a look on this example:

`http://localhost:8080/Insert/info?name=antonio &surname=sapuppo&username=anto82`

To the address `http://localhost:8080/Insert/info` the following parameters are given:

- Name = Antonio
- Surname = Sapuppo
- Username = anto82

The connection can be specified in the following steps:

- Setup: all the parameters have to be set before they are sent
- Connection: A connection is established, the parameter has been sent and the client is waiting for a response
- Communication ended: it is the last operation which has to be done closing the HTTP communication

HTTP supports a fixed set of methods (GET, PUT, POST, etc.). Spider uses only two of them: GET and POST. Almost all of requests take the form of the GET except of uploading pictures to the server. In this case the application uses a POST. The difference between the two kinds of requests is that the GET form enables sending only a restricted amount of information, while the POST form allows for sending more information to the server which does not have to be textual data, but can even take a form of files and binary data.

Now, we are going to describe the way Spider uses these requests to download and upload data from and to the server. Example code 2 shows how to open an HTTP connection using a GET method.

Example code 2 Http Connection - GET METHOD

```
1: HttpConnection http = (HttpConnection) Connector.open("url")
2: http.setRequestMethod(HttpConnection.GET)
3: if http.setRequestMethod() == HttpConnection.HTTP_OK then
4:   byte [] servletData = null
5:   InputStream input = http.openInputStream()
6:   int length = (int) http.getLength()
7:   if length > 0 then
8:     servletData = new byte [length]
9:     input.read(servletData)
10:    String data = new String(servletData)
11:    {processing data}
12:   end if
13: end if
```

In line 1 an HTTP connection is opened. This example shows the way the data are retrieved from the server. In the next line the kind of connection that has to be established is set. After verifying that the *Connection* was established correctly (line 3), an *InputStream* is opened to read the response of the server (lines 5-10). Then the obtained response will be used for further processing (line 11). When the resources are not needed anymore they have to be closed.

Example code 3 shows an example of how to use the POST request to upload an image to the server.

Example code 3 HTTP connection - POST METHOD

```
1: HttpConnection hc = (HttpConnection) Connector.open("url")
2: hc.setRequestMethod(HttpConnection.POST)
3: OutputStream out = hc.openOutputStream()
4: out.write(stringImage.getBytes())
5: if out != null then
6:   out.close()
7: end if
8: if hc != null then
9:   hc.close()
10: end if
```

In first line an HTTP connection is established and in the next line the modality of request is set. In this case a POST request is set. After that an *OutputStream* object is created and data are being sent to the server (line 4). When the *OutputStream* and the *HttpConnection* are not needed anymore they are closed (lines 5-9).

4.3.2 Server responses

The previous sections of this chapter regarded the way clients establish an HTTP connection and send requests to the server. The following section in turn presents in details the way the server gets the clients requests, executes queries on the database, receives its results and sends them back to the clients.

Example code 4 and 5 show an example of the way the server responses on the client's requests.

Example code 4 *HttpServlet* (*HttpServletRequest* request, *HttpServletResponse* response)

```
1: String name = request.getParameter("Name")
2: String surname = request.getParameter("Surname")
3: String user = request.getParameter("Username")
```

Example code 4 takes two parameters as an input:

1. *HttpServletRequest* request: this parameter is used to get the data sent by the client through the servlet
2. *HttpServletResponse* response: this parameter is used to send back the response obtained from the server to the client

In the lines 1-3, the parameters sent by the client are stored in different *String* variables. For example if the client has attached in the url *?Username=anto82*, now the value *anto82* is stored in the variable *user*. This is the way the servlet stores the data received from the clients.

Example code 5 presents the behaviour of servlet in case the client has set a GET request.

Example code 5 *HttpServlet* (*HttpServletRequest* request, *HttpServletResponse* response)

```
1: PrintWriter out = response.getWriter()
2: response.setContentType("text/plain")
3: String responseString
4: if Insert, delete or update queries then
5:   ResultSet rs = sta.executeUpdate("Query") {Insert, delete and update queries}
6:   responseString = "response of the server";
7: else
8:   ResultSet rs = sta.executeQuery("Query") {Only select query}
9:   responseString = buildAnswerDB(rs)
10: end if
11: out.print(responseString)
```

In the line 1 an object *java.io.PrintWriter* is obtained, which is used by the server to respond to the client. In the next line a type of response is set. In this example the response will take a form of a text. If the server has to execute an insert, delete or update query, it uses the *updateQuery* method of the *Statement* object. Otherwise, the server uses the *executeQuery* method and the response obtained from the database is further processed (lines 8-9). This response is given as an input parameter to the *buildAnswerDB* method which processes it and returns a *String* type containing the response. At the end this response is sent to the client (line 11).

The *buildAnswerDB* method is shown in Example code 6.

Example code 6 buildAnswerDB method (*ResultSet* rs)

```

1: String fields
2: ResultSetMetaData metadata = rs.getMetaData()
3: int totColumn = metadata.getColumnCount()
4: while rs.next() do
5:   for i = 0 to totColumn do
6:     fields = fields+rs.getString((i+1))+“|“
7:   end for
8: end while
9: return fields

```

Example code 6 takes as an input the *ResultSet* object which contains the response of the database. In the line 1 a *ResultSetMetaData* is initialized. This object is used to get information regarding types and properties of the table columns present in the *ResultSet* object. In the next line the number of columns is calculated. In lines 4 to 9 the response of the database is stored in a *String* type where obtained values are separated by a special character "|".

Example code 7 presents the behaviour of the server in case the client uploads an image using a POST method.

Example code 7 HttpServlet (HttpServletRequest request, HttpServletResponse response)

```

1: InputStream in = request.getInputStream()
2: BufferedReader r = new BufferedReader(new InputStreamReader(in))
3: StringBuffer buf = new StringBuffer()
4: String line
5: while line = r.readLine() != null do
6:   buf.append(line)
7: end while
8: String imageString = buf.toString()
9: sta.executeUpdate("Query")

```

In Example code 7 an *InputStream* and a *BufferedReader* are created (lines 1-2) in order to receive the image which the client uploads. The buffer will be filled with the bytes of the image (lines 5-7), and then the bytes of the image will be stored in the *imageString* variable (line 8).

Next, the server sends the *imageString* data to the database using the *java.sql.Statement* object (line 9) and then sends a positive answer to the client.

These are the principal methods of the client/server communication used by the Spider application.

Chapter 5

Client side: Internet connectivity

The previous section of the following project was devoted to the server part of the application. Moreover, it was explained how clients and server manage to connect and exchange data. This section in turn explores all possibilities that the Spider application provides for clients.

First, a closer look will be taken at the RMS package, as Spider is highly dependent on the possibility to store and retrieve data from the internal memory of the device. Next, the actions which clients can perform will be described. A pseudo code of these actions will be presented and explained to the reader in a comprehensible way. Moreover for each client's action the server's behaviour is going to be discussed.

5.1 RMS - Record Management System

The Mobile Information Device Profile provides a mechanism for MIDlets to persistently store data and later retrieve it [19]. This constant storage mechanism is referred to as the Record Management System (RMS). It is patterned on a simple record-oriented database. RMS constitutes a crucial resource for Spider application because first, it needs to store data even if the application ends or the mobile device is turned off, and second, it needs to retrieve data for other application executions.

Figure 5.1 presents the Record Management System.

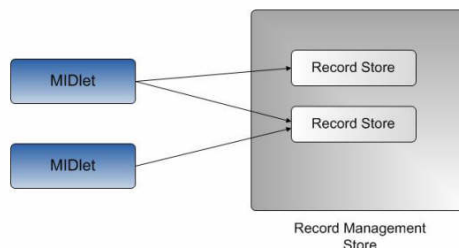


Figure 5.1: Record Management System

As seen in the Figure 5.1, the basic storage mechanism used by the RMS is called a Record store. Each record store consists of a set of records that will stay persistent during multiple invocations of a MIDlet. To be more precise, a Record store is a set of records, while a record is basically a byte array of arbitrary data. The size of the byte array is not given - it may vary for each record. The RMS does not understand the contents of a record and does not place any restrictions on what goes into the record. The platform is in charge of maintaining the integrity of the MIDlet's record stores throughout the regular use of the platform (reboots,

battery changes, etc.).

The RMS assigns to each record a unique identifier (`RecordId`) which is valid for the lifetime of the record store [18]. To the record created as the first one, it will attribute the `RecordId` value of 1; the secondly created record will be given a `RecordId` value of 2. This pattern is valid for all of the following records (Figure 5.2).

| RecordId | Data |
|----------|--------|
| 1 | Data 1 |
| 2 | Data 2 |
| 3 | Data 3 |
| ... | ... |
| N | Data N |

Figure 5.2: Storage mechanism of the RMS

The use of record stores provides certain advantages. First, it allows the MIDlet to remove and add records, and it allows MIDlets within the same MIDlet suite to access shared records. Recent versions of MIDP (2.0 and subsequent ones) provide the possibility of sharing a record store between all applications on the phone. Next, reading and writing to records are guaranteed to be atomic with no possibility of data corruption.

One can open a record store by the `openRecordStore` (*String recordStoreName, boolean createIfNecessary*) method. The name is the record store identifier. If no record store of that name exists, the boolean indicates whether it should be created or not. The record store can be closed by the `closeRecordStore` method.

5.2 Client actions

Client actions are divided into two groups referred to as initialization actions and mission actions. The former group comprises the methods to initialize the application by registration or various forms of log-in, while the latter one includes all possible actions the application provides when a user is already logged in.

5.2.1 Initialization actions

Figure 5.3-a shows the first screen displayed when the application starts.



Figure 5.3: Initialization actions

After the user chooses the *Intro* command the following screen presented in the Figure 5.3-b will be displayed. At this point, by clicking on the *Options* command, the user can make a choice between possible actions (Figure 5.3-c):

- Login
- Register a new user
- Delete a user
- Log in as another user

These actions are going to be explained below.

Register a new user

This action is used when a user would like to register in the application and create his profile. When the user clicks on the *Register a new user* command, the screenshot on Figure 5.4-a will appear.



Figure 5.4: Registration form

The application shows the registration form which has fulfilled by the user (Figure 5.4-b and Figure 5.4-c). Some of these fields are obligatory while fulfillment of others is optional.

The obligatory fields are the ones that constitute components of a business card of the user. These are the following:

- Username
- Password
- Home city
- Sex
- Country
- Picture

The application is tested on *Nokia N95* which supports camera technology, however Spider could also work on some other phone models which are not equipped with a camera. Therefore, if the picture is not inserted, a default picture will be displayed.

Figures 5.5-a and 5.5-b present a list of obligatory and optional parameters that have been already inserted by the registration process. The way in which these parameters were inserted shall be described below.

When a user clicks on a particular field to insert an appropriate information, another screen appears. For instance when the *Username* field is clicked on, the screen presented in Figure 5.5-c appears. Here a user can insert a chosen name and store it using the *Save* command.



Figure 5.5: Fullfilment of the registration form

What is important, these input parameters can be inserted in various ways. In the above mentioned case of the Username field, a form of text field is required. Other fields require inserting data in the form of date (for instance in Date of birth field - Figure 5.6-a). Figure 5.6-b in turn shows the Choice Exclusive input parameter, where the user may choose between two mutually exclusive options (in this example marking the gender), and Figure 5.6-c shows the Choice Multiple input parameters, where the user can mark as many options, as he wants (in this case marking his languages).



Figure 5.6: Exemplary fields requiring different data input

The application is capable of recognizing whether the type of inserted data is the appropriate one for a relevant field because it receives the information regarding rules and requirements for each field of the registration from the server. The details concerning the way in which client receives this information from the server are given in previous chapter.

After marking the selected fields, the user has to click on the *Save* command to store these selections.

Figures 5.7-a and 5.7-b present how to insert a picture into the application. After clicking on the field *Picture* on the screen presented in the Figure 5.5-b, the camera resource of the mobile phone will appear. The user can take a picture (Figure 5.7-a) and then he can store it by clicking on the *Save* command or, if he doesn't like it, he can decide to take another picture by clicking on the *Retry* command (Figure 5.7-b).

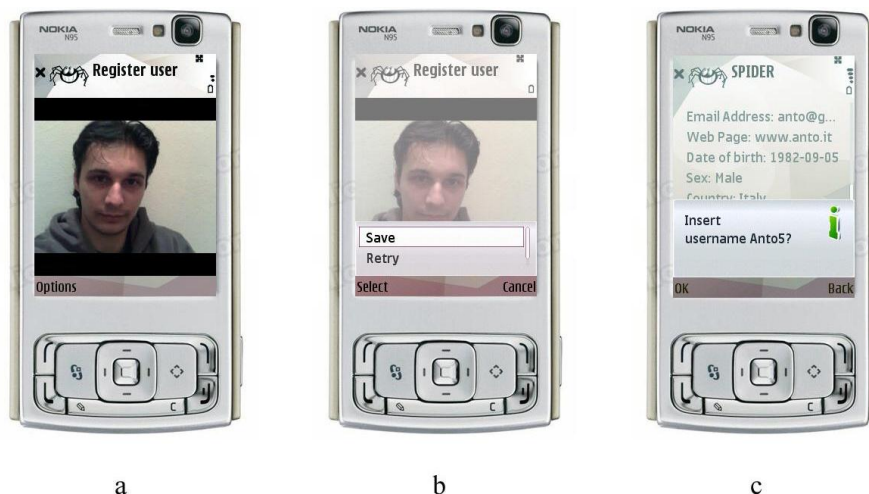


Figure 5.7: Adding picture to the user profile

Now that all the data has been inserted, the user can click on the *Register* command in the screen shown in the Figure 5.5-a. Then the screen presented in Figure 5.7-c will appear.

If the user has made some mistake by inserting data, the application will inform him about the kind of error. Figure 5.8 shows all possible errors that a user can do when inserting his parameters:

- Username, Password, Date of birth, Home city are an empty fields
- Password should consist of at least 6 characters
- Password and Re-Write Password fields were fulfilled in different ways

In these cases, the application makes the user insert all data correctly.



Figure 5.8: Error messages

Another error may occur when a user chooses a username which has already been taken by someone else. In this case the application will return an error alert, as it is shown in Figure 5.9.

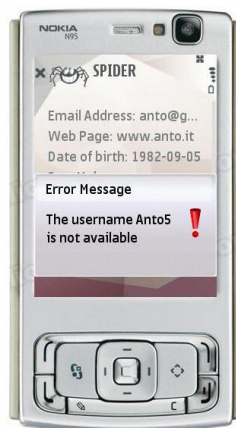


Figure 5.9: Error message - typed username is not available

The user has to choose another username; in this example the user typed *Anto5982* as his new username.



Figure 5.10: Sending request to the server - Successful registration

Figure 5.10-a shows the request for a confirmation of inserted parameters. After the confirmation has been received, the bar inserting line will be displayed, which means that the profile is being inserted into the application (Figure 5.10-b). Figure 5.10-c in turn shows that the data have been inserted correctly.

After processing the registration, the application will display the business card of the user. It can include either the picture taken by the user or a default picture inserted by the application (Figures 5.11-a and 5.11-b).

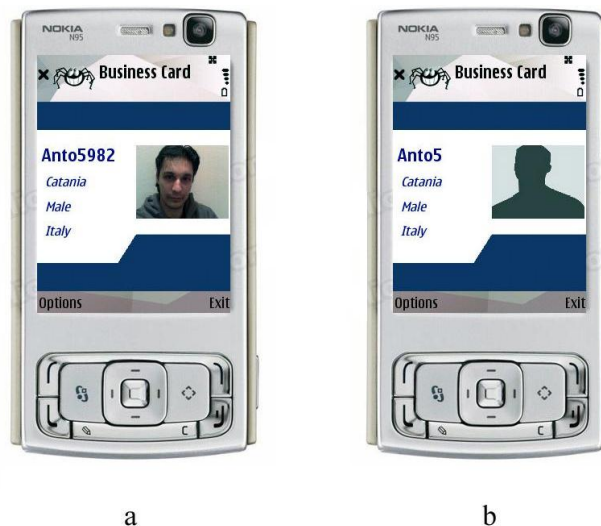


Figure 5.11: Business card

Below the pseudo code of this action is presented.

Example code 8 Register a new user

```
1: Vector v = Connects to the Server(url)
2: for i = 0 to v.size() do
3:   Scrolls the vector and creates the appropriate Object for the Registration form
4: end for
5: { The user has to type the parameters to create his profile }
6: Vector parametersInserted = Stores the selections of the user
7: Boolean checkValue = Checks if parametersInserted are correct
8: if !checkValue then
9:   Shows an error alert
10: else
11:   Vector businessCard = extracts business card from parametersInserted
12:   if image_object != null then
13:     Adds to the parametersInserted the image of the user
14:     Adds to the businessCard the image of the user
15:   else
16:     Adds to the businessCard the default image
17:   end if
18:   Sends parametersInserted to the server(url)
19:   if username already exists then
20:     Shows an error alert
21:   else
22:     Stores the businessCard in the local memory of the device
23:     Shows a confirmation alert
24:   end if
25: end if
```

The following explanation should facilitate understanding of the pseudo code of the *Register a new user* method.

At the beginning the application connects to the server using an internet access and it receives the response of the server. Further, the response is converted into a Vector type. The application elaborates this vector and creates a list of the fields shown in Figure 5.4-a and Figure 5.4-b (lines 1-4).

Once the fields have been fulfilled by the user (line 5), these data are sent to the server (line 18). In case these fields are not typed correctly the application returns an error message (line 9).

If the user has taken a picture (line 12), this picture will be added to his business card (line 14) and sent to the server. Otherwise a default picture will be a part of the business card (line 16). The default picture will only be stored in the local memory and it will not be sent to the server.

If the username chosen by the user already exists, the application receives a negative response from the server and shows an error message (line 20) which informs that the chosen username is no longer available. Otherwise, the application receives a positive response from the server and it stores the business card into its local memory (line 22). Finally, a confirmation alert is shown (line 23) which informs that a new user has been created.

Example code 8: How does the server deal with requests of clients? On this registration form client communicates with the server only two times. The first time refers to the line 1 of the pseudo code where the client requests the server for the profile form of the user. When the server receives this request, it just returns the fields for building the profile which have to be filled by the user.

The server uses a Select query in the tables which have headings beginning with character 'A'. The character 'A' identifies in the database tables referring to the profile of the user. Specifically, these tables are AATXFNPerson, ABDATSDate, ACCGELSex, ADCGPLCountry, AECGMSLanguages, AFCGMSInterests, AGPHOLPicture. More details concerning these tables were given in the previous chapter.

The second time when client communicates with the server refers to the 18th line of the pseudo code. At this point the client asks the server to create a new user using the typed values

sent as parameters.

The server receives all values which were typed or selected by the user, and using a `Select` query it checks in the table `AATXFPerson` of the database if the username chosen by the new user already exists. If this specific username has been already chosen by another user, a negative answer is given back to the client. Otherwise the server inserts each value of the profile to the relevant table of the database. Then a positive answer is sent to the client to inform him that the data provided by the user have been inserted correctly into the tables of the database.

Delete user

This action is used when a user wants to be deleted from the application. In the first screen of the application (Figure 5.3-b), if the user clicks on the *Username* field, a list which contains all usernames stored in the local memory of the mobile will be displayed (Figure 5.12-a). Here the user can choose the username that he would like to delete - in this example the username is *Anto5* (Figure 5.12-b). Then the user should click on the *Options* command and next choose a *Delete user* option from the displayed menu (Figure 5.12-c).



Figure 5.12: Delete user

Once the user clicks on the *Delete User* command, the application checks if the username and the corresponding password typed by the user are correct. If they do not match, an error alert is shown on the screen (Figure 5.13).



Figure 5.13: Error message - username and password do not match

If the username and the password match, the application shows an information alert which asks if the username should be deleted only from the local memory of the mobile phone or

also from the database which is placed in the server side (Figure 5.14-a). If the user decides to delete the username *Anto5* only from the local memory, *Anto5* is still an active username which can be used through another mobile phone or through the web site.

Otherwise, if the user decides to delete the username also from the server, the username *Anto5* will not be available anymore (Figures 5.14-b and 5.14-c).



Figure 5.14: Sending request to the server - Successful deletion

The pseudo code given below should facilitate understanding of the *Delete user* method.

Example code 9 Delete user

```

1: {Selects the username and types the corresponding password}
2: if Username AND Password match then
3:   if Deleting from the server then
4:     Connects to the server(url)
5:   end if
6:   Deletes the chosen username from the local memory
7:   Shows a confirmation alert
8: else
9:   Shows an error alert
10: end if

```

In this piece of pseudo code the application first checks into its local memory if the username and the typed password match (line 2). If they do not, an error alert is shown on the display of the device (line 9). If the typed values match and the user chooses to delete the username only from the local memory, the application doesn't need to connect to internet and it deletes the business card and all data regarding this user from the local memory of the device (line 6). If the user chooses the other option, the application connects with the server using an internet access and deletes the data regarding this username both from the database (line 4) and from the local memory (line 6). In both cases a confirmation alert is shown on the display (line 7).

Example code 9: How does the server deal with requests of clients? In this client action the communication with the server is established only once which is shown in the 4th line of the pseudo code. The client requests the server to delete a user from the database and all his interactions with other users. The server deletes the user using a Delete query in the following tables of the database:

- AATXFNPerson, ABDATSDate, ACCGELSex, ADCGPLCountry, AECGMSLanguages, AFCGMSInterests, AGPHOLPicture to delete the profile of the user
- CASMSSMessage to delete the messages sent by and received from the user
- BATABLFriends to delete the friends of the user

- BBTABLBlockedFriends to delete the invitations sent by and received from the user

After this process the server sends back a positive response to the client to inform that the user and all his interactions with other users have been deleted.

Login as another user

This action is used to insert to the local memory of the mobile phone the business card of a user. Let us assume that a user had already registered using the web site or another mobile phone. Using this action, the user does not have to create a new username but he can use the one that he has created previously. In the first screen of the application the user clicks on the *Login as another user* command (Figure 5.15-a). Then a new screen appears and he types the username and the corresponding password (Figure 5.15-b). The application connects to the server (Figure 5.15-c), and if the username and the password match, the business card of the typed username will be stored into the local memory of the mobile phone. Otherwise an error alert will be shown (Figure 5.13).



Figure 5.15: Login as another user

Example code 10 presents the pseudo code of this action.

Example code 10 Login as another user

```

1: {Types username and password}
2: Vector v = Connects to the server(url)
3: if Username AND Password match then
4:   for i = 0 to v.size() do
5:     Scrolls the vector and obtains the business card
6:   end for
7:   Stores the business card into the local memory
8:   Shows a confirmation alert
9: else
10:  Shows an alert error
11: end if

```

The first part of the pseudo code handles the connection with the server to send a request (line 2). If the username and the password typed by the user do not match, an error alert is shown (line 10) since the server sends back a negative response. Otherwise the response of the server is converted into a vector object. The response in the form of a vector is elaborated by the application into a business card which is then stored in the local memory of the device (lines 4-7).

Example code 10: How does the server deal with requests of clients? In this method only one connection is established from the client to the server. The 2st line of the pseudo code refers to their communication. The server gets two parameters: username and password as an

input. Using a Select query it checks if these two values match in the AATXFNPerson table. If they do not match, a negative response is sent back to the client. If in turn the inserted username and password match, the server returns the business card of the user.

The business card is returned using select queries in the fields of the tables which headings contain 'A' as the first character ('A' represents the identify value of the profile) and 'L' as the 6th character ('L' means Local due to the fact that this value has to be stored on the local memory of the device) and in the tables which headings contain 'A' as the first character and 'N' as the 6th character. However, the values obtained from the tables which headings contain 'N' as the 6th character have to be filtered one more time. Inside these tables, only the fields which contain 'L' as the second character will be subject to another select query.

To sum up, the Select query will be executed in the following fields:

- ALUsername, ALHome City of the AATXFNPerson table
- Male and Female of the ACCGELSex table
- Country of the ADCGPLCountry table
- Picture of the AFPHOLPicture table

The business card of the user is composed of the select query results in these fields. More details were given in the previous chapter.

Login

The user chooses this action when he would like to log in into the application. When a user uses the *Registration* action or the *Login as another user* action, logging into the application is done automatically and the business card is stored in the local memory of the mobile phone.

From the second access, he can log in using the *Login* command due to the fact that his business card is stored in the internal database of the device. First of all, he has to select the username which should log in into the application (Figure 5.16-a). Next, the user has to type the password (Figure 5.16-b) and click on the *Login* command (Figure 5.16-c). If the username and the password match, the business card of the username will be shown. Otherwise an error alert will be displayed (Figure 5.13).



Figure 5.16: Login

The pseudo code of the login method is presented below.

Example code 11 Login

```
1: {Selects the username and types the corresponding password}  
2: if Username AND Password match then  
3:   Shows business card  
4: else  
5:   Shows an alert error  
6: end if
```

The pseudo code of the Login method is really easy to understand. There is only one concept that has to be marked in this method. The login method is used only with usernames which are stored in the local memory of the device, which means that a user registered himself using this particular device or that he downloaded his already existing username into the device. If a username is a valid one but is not stored in the local memory, this username will not appear in the list of the usernames available under the login command in this specific device. No connection with the server is established for this action.

5.2.2 Mission actions

After having discussed the possible ways in which a user can register or log in into the application, the following section will focus on all available actions the user can take from that moment. These actions become available after the application displays the business card of a particular user. The business card of the user is shown in three cases:

1. After the *Registration* process
2. After using the *Login as another user* command
3. Each time that a user uses the *Login* command

Figure 5.17-a shows the business card of the username *Anto5982* and Figure 5.17-b shows the possible actions that the user can perform after logging in into the application. Once the user has logged in into the application, he can perform the following actions:

- My virtual world
- My profile
- My friends
- Retrieve news
- Search new contacts
- Synch with the server

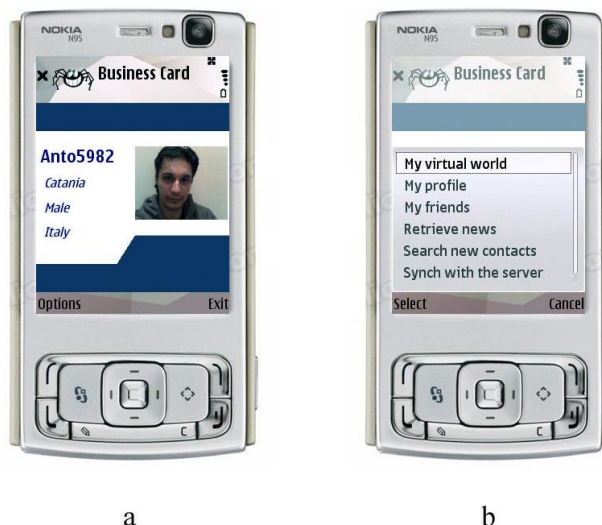


Figure 5.17: Business card and the mission actions menu

All of the above written actions are going to be described below. However, the *My virtual world* command unlike the rest of actions does not use an internet connection but instead requires a Bluetooth resource, and therefore will be described in the next chapter of the thesis.

My profile

Using *My profile* command the user is able to check or change his own profile. After the *My profile* option is chosen (Figure 5.17-b), The full profile of the user logged in the application will be shown (Figure 5.18).



Figure 5.18: Profile of the user

If the user clicks on the *Options* command, he can choose between 2 commands: *My picture* and *Update my profile*. If the user clicks on *My picture* (Figure 5.19-a), his picture will be resized to fit the whole screen of the mobile phone (Figure 5.19-b).

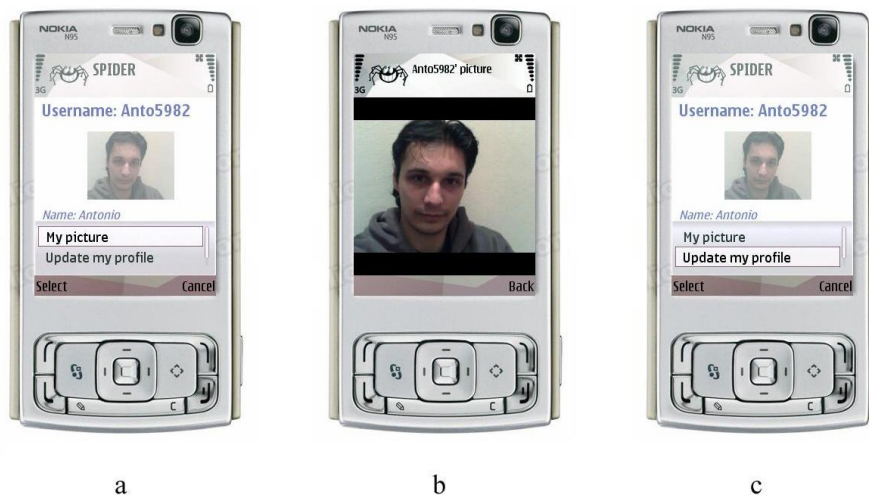


Figure 5.19: My picture and Update my profile commands

If the user clicks on the *Update my profile* command (Figure 5.19-c), he has the possibility to change data given in his profile. A list of data included in the profile of the user is shown in Figure 5.20-a and Figure 5.20-b. The user can click on fields he would like to modify and insert new information. In this example the user clicks in *Picture* field and he takes a new picture (Figure 5.20-c).



Figure 5.20: Update my profile form

Now the user should click on the *Update* command from Figure 5.20-a so that the inserted data can be updated in the application. The application will ask the user for a confirmation (Figure 5.21-a), then display a bar uploading line (Figure 5.21-b) and finally show the information alert stating that the profile has been updated (Figure 5.21-c).



Figure 5.21: Sending request to the server - Successful update

After receiving the confirmation alert the application displays on the business card (Figure 5.22). As it is possible to see the picture has been updated.



Figure 5.22: Business card

The pseudo code of the *My profile* action is presented below (Example code 12).

Example code 12 My profile

```
1: Vector v = Connects to the server(url)
2: Canvas canvasProfile = new Canvas() { Creates a Canvas object to visualize the profile of the user }
3: for i = 0 to v.size() do
4:   Scrolls the vector and fills the canvasProfile with fields (username, name, etc.) and their corresponding
   values (Anto5982, Antonio, etc.)
5: end for
6: Adds two Commands (My picture and Update my profile) to the canvasProfile and shows the canvasProfile
   object on the screen
7: if My picture command is activated then
8:   Shows the picture on the full screen
9: else if Update my profile command is activated then
10:  for i = 0 to v.size() do
11:    Scrolls the vector and creates the appropriate object for the Update form
12:    Inserts into each object the corresponding value which the user has typed before
13:  end for
14:  {The user may type new parameters to update his profile}
15:  Boolean access = Checks if the user's parameters were inserted correctly
16:  if access then
17:    Sends data to the server(url)
18:    Updates the business card in the local memory
19:    Shows a confirmation alert
20:    Shows the business card of the user
21:  else
22:    Shows an error alert
23:  end if
24: end if
```

To perform this action the application needs an internet access as it is shown in the line 1 of the pseudo code. The application gets the server response and converts it in a vector object. In the next lines (2-5), the application initializes a canvas object which is used to show the full profile of the user on the screen. Specifically, the full profile of the user is received as a response of the server. The application processes this response into a vector object which is then elaborated to fill the canvas object.

If the user chooses to view his picture, the canvas will be refreshed and the picture of the user will be resized to the width and the height of the screen (lines 7-8).

If the user chooses to update his profile, the application elaborates the vector which contains the full profile of the user and creates a form object called update form similar to the registration form. The only difference between them is that the registration form is composed only of the fields of the profile and the update form is composed of the fields of the profile and values typed previously (lines 9-13). Once that values of some fields have been modified, the application checks if the data have been inserted correctly (line 16). If not, the application shows an error alert and makes the user insert the data in an appropriate manner (line 22). Otherwise the data are sent to the server and if some alterations have been done in the business card fields, these changes will be updated in the local memory of the device and a confirmation alert will be shown (lines 17-19). After several seconds the application will display the updated business card of the user (line 20).

Example code 12: How does the server deal with requests of clients? In this method the client connects with the server at two points referring to the 1st and the 17th line of the pseudo code.

In the first line the client requests the server to send the full profile of a specific user. The server gets the username as an input parameter. Further on, using this value the server executes select queries in the tables of the database which headings contain 'A' as first character, since this character identifies the profile of the user in the database (the same process as in the Register a new user action). The fields of the table and in this case also their corresponding values obtained using the select queries will be returned to the client.

In the line 17 the client asks the server to update the profile of a specific user. The server gets the username of the user and the values which have to be updated in the relevant tables as parameters. Using Update queries each value typed by the user is going to be updated in the relevant table of the database. Afterwards a positive response will be sent back to the client.

Search new contacts

The user performs this action, when he is looking for new contacts. After clicking on the *Search new contacts* command (Figure 5.23-a) another screen containing several search categories will appear (Figure 5.23-b).

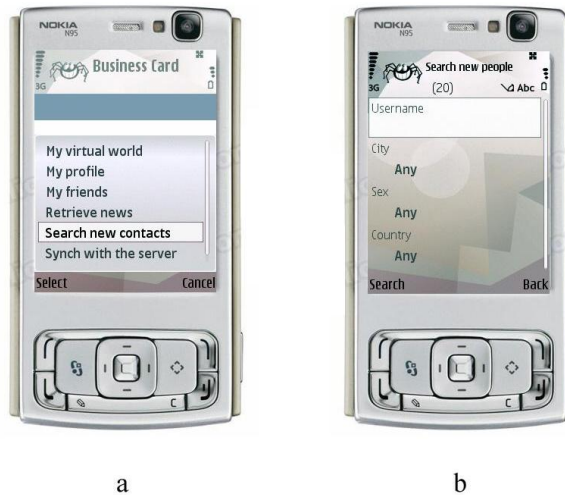


Figure 5.23: Search new contacts form

Thus contacts can be sought on the basis of the following filters criteria:

- Username: since it is a text field, the user can insert some text. The application will discard all usernames that do not contain all the characters inserted by the user
- City: it contains all the cities that have been inserted into the database by users during the registration phase (Figure 5.24-a)
- Sex: it contains male and female values(Figure 5.24-b)
- Country: it contains all the countries which have been inserted into the database by users during the registration phase (Figure 5.24-c)

The last three fields also contain an element called *Any*. If this element is selected, the corresponding field will not constitute a part of the research.

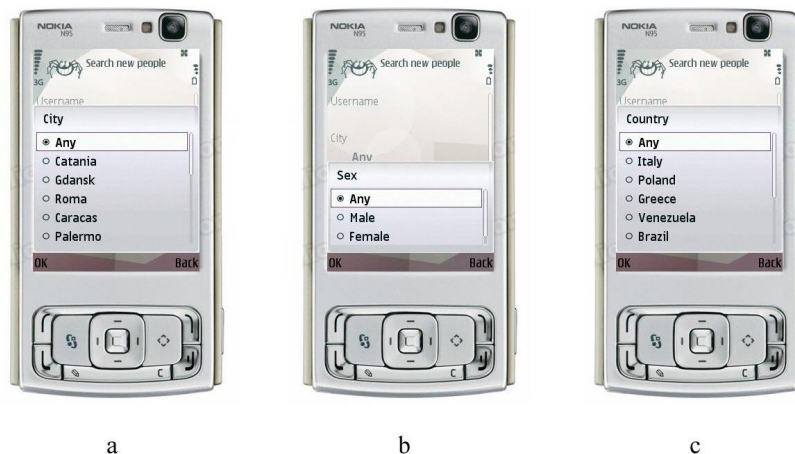


Figure 5.24: Fullfilment of the Search new contacts form

In this example the following filters criteria have been chosen by the user (Figure 5.25):

- Username: ka
- City: Any
- Country: Poland
- Sex: Female



Figure 5.25: Example of the search results with the filters criteria applied

After choosing filters for searching contacts, the user clicks on the *Search* command and a list of people who fulfill the inserted filters criteria appears. In this example only one user meets these requirements. When the user clicks on this element, he can see either the full profile of the found person or only his or her business card, depending if they are friends or not. In this example, *Karola83* was not a friend of *Anto5982* and hence only a business card of *Karola83* is displayed on the screen (Figure 5.25-c). Now the user can send a message or an invitation to *Karola83*. The user has to click on the *Options* command and then he can choose the action to perform (Figure 5.26).



Figure 5.26: Send message action

In this example, the user can send a message to the found contact. In Figure 5.26-a, after he clicks on the *Send sms* command, a new screen will appear which gives the opportunity to type the message. After typing the message, the user has to click the *Send* command (Figure 5.26-b). The application shows an information alert asking the user for a confirmation to send the

message to the chosen username (Figure 5.26-c). If the username clicks on the *Ok* command, the message will be sent (Figure 5.27-a).

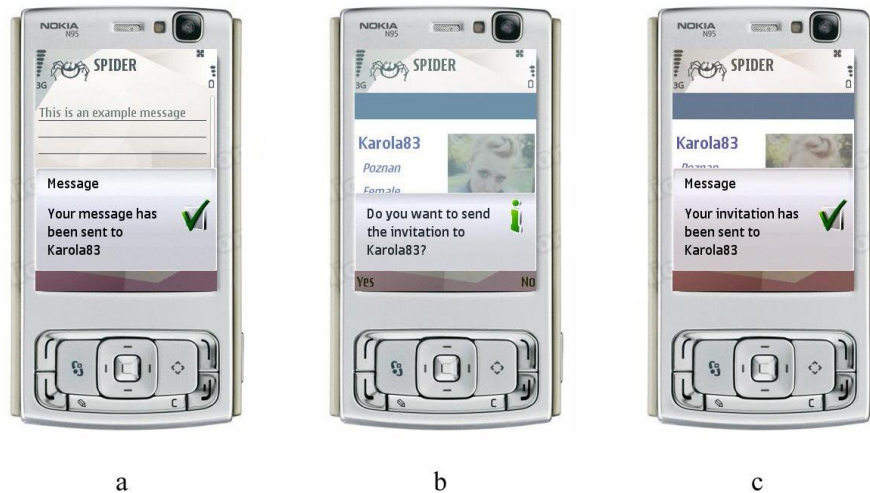


Figure 5.27: Send invitation action

The user can also send an invitation to *Karola83* to be friends. The status of friends gives users the opportunity to view the full profile of the friend and to check friends list of a friend. Let us take an example: if *Anto5982* sends an invitation to *Karola83* and she accepts the invitation; *Anto5982* is able to view the full profile of *Karola83* and the list of friends of *Karola83*. There are additional advantages to be friends which will be explained in the next chapter of the project, due to the fact that they regard the Bluetooth part. To send an invitation, from the screenshot of Figure 5.25-a, the user has to click on *Send Invitation* command. The application will show an information alert (Figure 5.27-b), and after the user clicks on the *Ok* command the invitation will be sent to the chosen username (Figure 5.27-c). The pseudo code of the Search new contacts action is shown below.

Example code 13 Search new contacts

```

1: Vector v = Connects to the server(url)
2: for i = 0 to v.size() do
3:   Inserts into the City field the cities present in the database and the Any value
4:   Inserts into the Country field the countries present in the database and the any value
5: end for
6: Inserts into the Sex field the Male, Female and Any values
7: {The user has to type the pseudo username and makes the selections on sex, country and city fields}
8: Vector e = Sends the user's selections to the server(url)
9: List foundPeople = new List()
10: for i = 0 to e.size() do
11:   String currentUsername = (String) e.elementAt(i)
12:   if pseudo_username.uqual(null) OR all characters typed in the pseudo_username field are included in
   the currentUsername then
13:     Adds to foundPeople currentUsername
14:   end if
15: end for
16: if foundPeople.size()=0 then
17:   Shows an information alert
18: else
19:   Shows foundPeople list
20:   if Clicks in one of the elements of foundPeople then
21:     Vector usernameChosen = Sends the name of the chosen username from the list of found people to
   the server(url)
22:     if Chosen username is not a friend then
23:       isNotFriend(usernameChosen)
24:     else
25:       isFriend(usernameChosen)
26:     end if
27:   end if
28: end if

```

At the beginning the application creates the form object which contains the filters that the user can use to find people which meet these requirements (line 1-6). After the user chooses the values of sex, city and country fields, the application returns a list of people which satisfies these requirements. If the user typed any character in the username field this list have to satisfy also the username requirements: it means that all characters inserted by the user have to be a part of usernames returned by the server (line 8-14).

If the list of found people is empty, it means that no username satisfies the requirements. In this case the application shows an information alert (line 17). Otherwise, if there are users that meet the inserted requirements, the list of found people will be shown on the screen (line 19). Now, the user can choose one username from the list (line 20). If the selected element is not a friend of the user, the application will invoke the *isNotFriend* method (line 23); otherwise it will invoke the *isFriend* method (line 25). These two methods will be described later on.

Example code 13: How does the server deal with requests of clients? In Search new contacts method, several connections are established between client and server. The first connection is referred to in the 1st line of the pseudo code where the client requests the server for the list of cities and countries inserted into the database. The server executes a select query in the ALHome City field of the AATXFNPerson table and in the Country field of the ADCGPLCountry table. The server returns to the client only distinct values of the obtained results.

Next connection is referred to in the 8th line, where the client asks the server for a list of users who satisfy his criteria in terms of the following parameters:

- City
- Country
- Sex

To perform this action the server gets as an input parameters regarding city, country and sex chosen by the user. Next it executes a select query in the Male and Female field of the ACCGELSex table, in the Home City field of the AATXFNPerson table and in the Country field of the ADCGPLCountry table. The list of usernames who satisfy all these requirements is sent back to the client.

Another client-server connection is referred to in the 21th line of the pseudo code. In this line the client requests the server to check if the sent usernames are the particular user's friends or not. If a user who satisfies all the search criteria is a friend of the user who inserted these criteria, the server will return his full profile. If a user found on the basis of these criteria is not a friend of the user who made a search, the server will return only his business card. To be more specific, the server gets as input value the username of the chosen user and it executes a select query on the BATABLFriends table. If this select query returns a negative answer, the business card is sent back to the client. The business card is returned using the same strategy which has been described in the *Login as another user* method. Otherwise, if the answer of the previous select query is positive, a full profile of the chosen username is returned using the same strategy shown on the *My profile* method.

The isNotFriend and isFriend methods This paragraph describes the two pseudo codes which are often invoked by the application: *isNotFriend* and *isFriend* methods.

Example code 14 shows the isNotFriend method.

When the *isNotFriend* method is invoked, the application takes as an input a vector which contains the business card of the chosen username.

Example code 14 *isNotFriend*(*Vector* usernameChosen)

```
1: { This method takes as an input the usernameChosen vector which contains the business card of the chosen username }
2: Canvas canvasBC = new Canvas() { Initializes a canvas object }
3: for i = 0 to usernameChosen.size() do
4:   Scrolls the vector and fills the canvasBC with fields and values regarding the business card
5: end for
6: Adds Send Sms and Send Invitation commands to the canvasBC
7: Shows canvasBC
8: if Send sms is activated then
9:   Sms is sent to the chosen username(url)
10:   Shows a confirmation alert
11: else if Send invitation is activated then
12:   Invitation is sent to the chosen username(url)
13:   Shows a confirmation alert
14: end if
```

A *Canvas* object is initialized (line 2) to show the business card of the chosen username. The business card is extracted from the vector to fill the *Canvas* object (lines 3-5).

Since in this case the chosen username is not a friend of the user, the following actions can be performed by the user:

- Send a sms (line 9)
- Send invitation (line 12)

In both cases, after sending the data to the server the application shows a confirmation alert (line 10, line 13).

Example code 15 shows the *isFriend* method.

Example code 15 *isFriend* (*Vector* usernameChosen)

```
1: { This method takes as an input the usernameChosen vector which contains the full profile of the chosen username }
2: Canvas canvasProfile = new Canvas() { Initializes a canvas object }
3: for i = 0 to usernameChosen.size() do
4:   Scrolls the vector and fills the canvasProfile with fields (username, name, etc.) and their corresponding values (Karola83, Karolina, etc.)
5: end for
6: Adds Picture and Send Sms commands to the canvasProfile
7: Shows canvasProfile
8: if Send Sms is activated then
9:   Sms is sent to the chosen username(url)
10:   Shows a confirmation alert
11: else if Picture is activated then
12:   Full screen picture of the chosen username is shown on the screen
13: end if
```

When the *isFriend* method is invoked, it takes as an input a vector which contains the full profile of the chosen username. This method is very similar to the previous one. Also in this method a *Canvas* object is initialized (line 2). The first difference between the two methods is that in case of *isNotFriend* method the *Canvas* object contains the business card of the chosen username, while in the *isFriend* method, the *Canvas* object will be filled with the full profile of the chosen username (line 3-5).

Another difference between the two methods is that in *isFriend* method the *Send Invitation* command is replaced by the *Picture* command (line 6). This is because the chosen username is a friend of the user, which means that the invitation command is not needed anymore. What is more, the user has the security permission to view the full profile of the chosen username, including enlarged picture.

The Send sms command is still available to provide the opportunity to send messages to the chosen username.

Example codes 14 and 15: How does the server deal with requests of clients? In these methods two connections are established with the server. The first one regards sending a sms to the chosen username (line 9 of both methods) and the second one regards sending an invitation to the chosen username (line 12 of Example code 14).

In the first connection, the server takes as an input three parameters:

1. the username of the sender
2. the username of the receiver
3. the text message

Afterwards the server executes an insert query with these parameters to the *CASMSSMessage* table and sends back a positive response to the client.

In the second connection, the server gets as an input the following parameters:

1. the username of the sender of the invitation
2. the username of the receiver of the invitation

Afterwards the server executes an insert query on the *BBTABLBlockedFriend* table. Also in this case the server sends a positive answer to the client.

My Friends

This action is performed by the user if he wants to see his friend's list. To become friends, a user needs to invite another user to join his group of acquaintances. Once the other user accepts the invitation, they become friends. Each user is added to the other user's list of friends.

From Figure 5.28-a, if the user clicks on the *My friends* command, the list of friends of the username *Anto5982* will be shown on the screen (5.28-b).

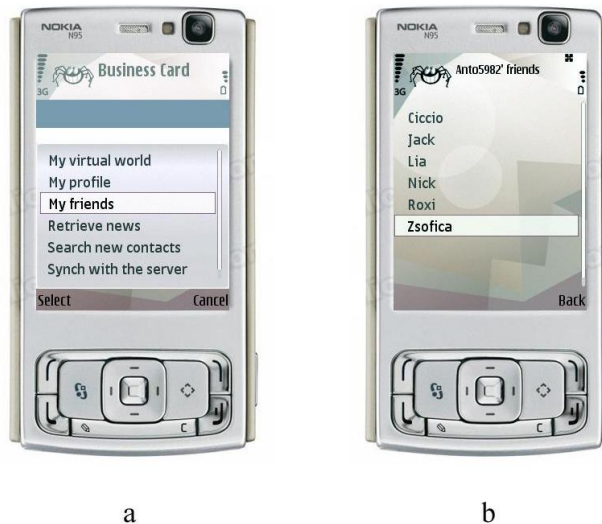


Figure 5.28: My friend command

Now if the user clicks on one of the elements of this list, the profile of the chosen person will be shown. In this example, the user clicks on the element *Zsofica* and the *Zsofica*'s profile is shown on the screen (Figure 5.29).



Figure 5.29: Viewing the full profile of a friend

Here the user can perform the following actions (Figure 5.30):

- Show the full screen picture of the chosen username
- Show the list of friends of the chosen username
- Send sms to the chosen username



Figure 5.30: Friend - possible actions

If the user clicks on the *Zsofica's picture* a full screen picture of this username will be shown. If the user clicks on the *Send sms* command, a short message can be sent to this username. These two actions had been already presented before.

If the user clicks on the *Zsofica's friends* command, the list of the friends of Zsofica will be displayed (5.31-a).



Figure 5.31: Zsofica 's friends

Now, if the user clicks on the *Nick* element from this list, the full profile of this username will be displayed, since *Nick* is also a friend of the user (Figure 5.31-b). If the user clicks on the *Ewe* element, only the business card of this username will be shown, since *Ewe* is not a friend of the user (Figure 5.31-c).

From the full profile screen the user can perform the following actions (Figure 5.32-a):

- Show the full screen picture (Picture)
- Send sms

From the business card screen the user can perform the following actions (Figure 5.32-b):

- Send Sms
- Send Invitation

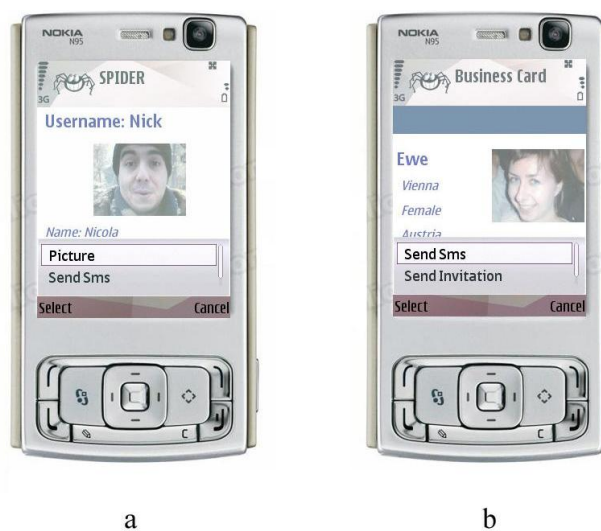


Figure 5.32: Friends and unknown people - possible actions

Below the pseudo code of *My Friend* action is shown.

Example code 16 My friends

```
1: Vector v = Connects to the server (url)
2: List listFriends = new List()
3: for i = 0 to v.size() do
4:   Adds to listFriends the list of friends received from the server
5: end for
6: Adds to listFriends the list of friends present in the local memory of the device that at the same time are
   not present in the list of friends received from the server
7: if listFriends.size() == 0 then
8:   Shows an information alert
9: else
10:  Shows listFriends
11:  if Clicks on one element of listFriends then
12:    Vector e=Sends the chosen username to the server(url)
13:    Canvas canvasFr=Shows the full profile of the chosen username(e)
14:    Adds 3 Commands (Picture, Send Sms, Friends) to canvasFr
15:    if Picture commad is activated then
16:      Full screen picture of the chosen username is shown on the screen
17:    else if Send sms is activated then
18:      Sms is sent to the chosen username(url)
19:    else if Friends is activated then
20:      List listFriend_friends=List of friends of the chosen username(url)
21:      if listFriend_friends.size() == 0 then
22:        Shows an information alert
23:      else
24:        if Clicks on one element of the listFriends then
25:          Vector elem = Sends the chosen element to the server(url)
26:          if chosen element is a friend then
27:            isFriend(elem)
28:          else
29:            isNotFriend(elem)
30:          end if
31:        end if
32:      end if
33:    end if
34:  end if
35: end if
```

Similarly to the previous methods, to perform this action an internet access is needed as already the first action is to connect with the server and request the list of friends of the user (lines 1-5). Afterwards the application adds to the list object the list of friends placed on the local memory of the device (line 6). Only usernames which were missing in the list obtained from the server will be added to the object list. Let us take an example to clarify this process. If the application gets from the server a list composed of Antonio, Karolina and Gianpa, and the list of local device is composed of Antonio, Karolina, Gianpa and Joao, the object list will be composed of Antonio, Karolina, Gianpa, Joao.

If the total list obtained is empty the application will show an information alert to the user (lines 7-8) which informs that the user doesn't have friends. Otherwise the application will show the list of friends of the user on the screen.

After showing on the display the list of friends, the user can choose one of the elements from the list. The chosen username will be sent to the server and afterwards the application will receive the server's response consisting of the full profile of the chosen element (lines 10-13). Now, the user can perform the following actions:

- See the full screen picture of the chosen username
- Send Sms
- Get the list of friends of the chosen username

The first two actions were explored previously, therefore this paragraph will focus on the third action: Get the list of friends of the chosen username.

The application sends to the server this request and as a response it receives the list of friends of the chosen username (line 20). If the user clicks on one of the elements of this list, two possible scenarios could appear:

1. The selected element is a friend of the user
2. The selected element is not a friend of the user

In the former case, the *isFriend* method will be invoked, while in the latter one the *isNotFriend* method will be called. The *isNotFriend* and *isFriend* methods were explained in the *Search new contacts* section and they refer to Example codes 14 and 15 respectively.

Example code 16: How does the server deal with requests of clients? In this piece of pseudo code several connections between client and server are established. The first connection is referred to in the 1st line where the client asks the server to send it the list of the user's friends. The server gets as an input the username of the user and it executes a select query in the BATABLFriends table. As a result of this operation a list of friends of the user is returned and sent back to the client.

The next connection between client and server is referred to in the 12th line of the pseudo code, where the client asks for the full profile of a friend of the user. The server gets as an input the username of the chosen friend and it executes a select query in a way that was described in the My Profile method. The full profile of the chosen friend is sent back to the client.

Another connection is referred to in the 20th line, where the client requests the server for a list of friends of a friend of the user. The server gets as an input the username of the user's friend and it executes a Select query on the BATABLFriends table. A list of friends of the chosen friend is obtained and sent back to the client.

Retrieve News

This action is performed in the following cases: when the user wants to read a received message or when he wants to accept or refuse the invitation sent to him by another user.

In the *Search new contacts* section the way in which messages and invitations are sent to other users was already explained. The way the *Retrieve news* command works is going to be presented.



Figure 5.33: My retrieve news action

From the screen presented in Figure 5.28-a the user may click on the *Retrieve news* command, and as a result the application will display a list which contains all messages received so far (Figure 5.33-a).

In this example a list of the news received by the user *Karola83* contains the username *Anto5982* mentioned twice, which means that *Karola83* received two messages from *Anto5982*. If the user clicks on the first element of the list, the message received from *Anto5982* will appear

on the screen (Figure 5.33-b). At this point the user can perform the following actions: *Reply* or *Cancel* the message (Figure 5.33-c).

If the user would like to respond to the sender of the message, he has to click on the *Reply* command and then the screen presented in Figure 5.34-a will appear. Here the user can type the message and click on the *Send* command (Figure 5.34-a). Before sending the message to the receiver the application asks the user for a confirmation, (Figure 5.34-b), and after getting it, the application displays an alert on the screen indicating that the message has been sent (Figure 5.34-c).



Figure 5.34: Reply message action

Another action that the user can perform from the screenshot seen in Figure 5.33-c is to cancel the message. If the user clicks on the *Cancel* command, an information alert will appear asking for the user's confirmation to delete the selected message (Figure 5.35-a). If the user clicks on the *OK* command, the message will be deleted and a confirmation alert will appear (Figure 5.35-b). Then, as it is seen in Figure 5.35-c, the message is not present in the list of retrieve news anymore.



Figure 5.35: Delete message action

Previously, the username *Anto5982* has sent an invitation to *Karola83*. This invitation is also placed in the list of retrieve news of *Karola83*. If the user (*Karola83*) clicks on *Anto5982* element of the list, the application recognizes that it is an invitation and asks the user if she would like to insert the sender of the invitation into the list of her friends (Figure 5.36-a).



Figure 5.36: Accepting or refusing an invitation

The user can accept or refuse the invitation by choosing *YES* or *NO* (Figure 5.36-a). After accepting or refusing an invitation by the user, an information alert is displayed (Figure 5.36-b) and the message will be deleted from the list of retrieve news (Figure 5.36-c).

Below the pseudo code of this action is presented (Example code 17).

Example code 17 Retrieve news

```

1: Vector v = Connects to the server(url)
2: for i = 0 to v.size() do
3:   Inserts into a list object the news of the user
4: end for
5: if Clicks on one element of the retrieve news list then
6:   Sends to the server the ID of the message(url)
7:   if ID.equals(message) then
8:     Form form = new Form()
9:     Shows the message on the form object
10:    Adds two commands (Reply,Cancel) to the form
11:    if Reply is activated then
12:      Sends a message to the sender(url)
13:    else if Cancel is activated then
14:      Deletes the message from the database(url)
15:      Deletes the message of the sender from the list of retrieve news
16:    end if
17:  else if ID.equals(invitation) then
18:    Shows an information alert with two commands (YES and NO) to ask wheter the user wants to add
    the new friend
19:    if YES is activated then
20:      Sends a request to the server stating that the receiver accepts the invitation(url)
21:      Adds the sender of the invitation to the list of the receiver's friends in the local memory of the
    device
22:    else if NO is activated then
23:      Sends a request to the server stating that the receiver refuses the invitation(url)
24:    end if
25:    Deletes the invitation of the sender from the list of retrieve news
26:  end if
27: end if

```

To perform this action an internet access is needed (line 1). The application requests the server for a list of news regarding a specific username. These news will be inserted into an object list (lines 2-4). Each time that the user clicks on an element of the retrieve news list, the application will request the server for the text of the message using the identification number of the chosen element (line 5-6). If the message is a text message, it will be shown on the screen of the mobile phone. Then the user can reply to this message or delete it (lines 11-16). Otherwise, if the message constitutes an invitation, the application will ask the user whether he wants to add the sender of the invitation to the list of his friends (line 18). If the user refuses

the invitation this request will be sent to the server and the message will be deleted from the list of news. If the user accepts the invitation in turn, the two sernames will become friends and this friendship will be stored in the local memory of the device. Additionally, the request will be sent to the server and the invitation message will be deleted from the list of the retrieve news of the user.

Example code 17: How does the server deal with requests of clients? In the pseudo code given above several connections are established between client and server. The first one is referred to in the 1st line, where the client asks the server for the list of messages received from other users. The server gets as an input the username of the user and executes a select query in the Receiver field of the CASMSSMessage table. It returns to the client the ID of the messages and the list of the senders' usernames.

When the user clicks on one element of the retrieve news list, this element could be either a message or an invitation - this request is shown in the 6th line. The server gets as input the ID of the element and it executes a select query in the idMessage of the CASMSSMessage table. If the result of the select query in the idMessage tells the server that the element is a text message, it will be returned to the client as text message. Otherwise this element will be returned to the client as an invitation message.

The next two connections established between client and server regard the reply for a message (line 12) and cancellation of a message (line 14). The reply message action is based on the same procedure as the process of sending a message and it was described in the Search new contacts method. In cancellation of a message action the client requests the server to delete a specific message from the database. The server gets as input the ID of the message which has to be deleted and it executes a delete query in the CASMSSMessage table. In both cases the server returns to the client a positive response.

The last two connections regard the accept (line 20) and refuse (line 23) invitation action. If the user refuses the invitation of another user, the server gets as input the usernames of the receiver and the sender as well as the idMessage of the invitation. Next, it executes a delete query in the BBTABLBlockedFriends table to delete the invitation. Then it executes an insert query to the CASMSSMessage to send an information message that the invitation has not been accepted and afterwards it executes another delete query on the CASMSSMessage to delete the invitation. In case the invitation is accepted, the server gets as input the same parameters and it performs the same actions as in the situation where the invitation was refused. The only difference is that obviously the message sent to the sender of the invitation will state that the invitation was accepted. Moreover, during this action the server performs two insert queries in the BATABLEFriends table. These insert queries will confirm the status of friendship between the sender and the receiver of the invitation and vice versa.

Synch with the server

This action is performed to synchronize the mobile phone with the server. Due to the fact that the user can interact with the server using different mobile phones and using the web site, different scenarios can appear:

- If the user adds new friends using the web site, they will be not present in the local memory of the mobile phone
- Using Bluetooth technology the user can add new friends which will be stored only in the local memory of the device and they will be not present in the server side
- Using the web site, the user can change some parameters of the business card, and these parameters will be not changed in the local memory of the device

Synch with the server action allows to avoid these problems as it synchronizes the server with the local memory of the device.

To perform this action the user has to click on *Synch with the server* command (Figure 5.37-a). An information alert will appear stating that the data are being updated (Figure 5.37-b) and at the end of the process a confirmation alert will inform that the data has been updated (Figure 5.37-c).

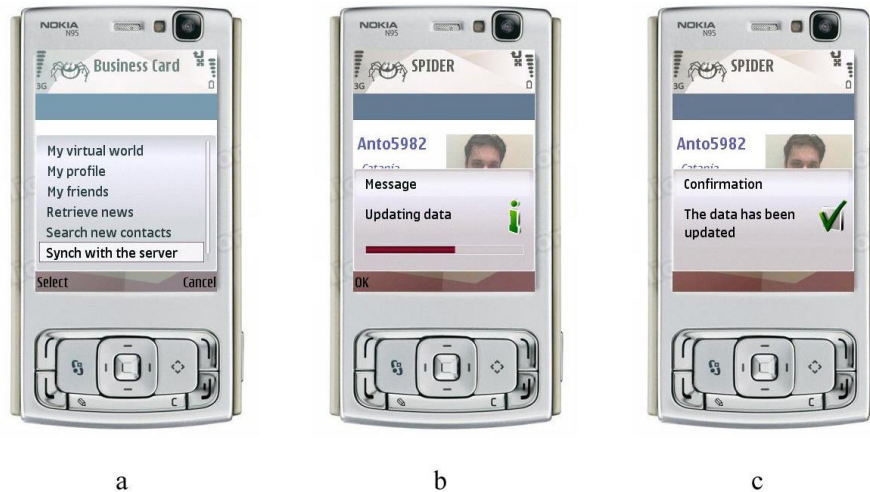


Figure 5.37: Synch with the server action

Below the pseudo code of this action is shown.

Example code 18 Synch with the server

```

1: Vector friendsDevice = List of friends present in the local memory of the device
2: Vector bcDevice=Business card present in the local memory of the device
3: Vector v = Connects to the server(url)
4: Vector friendsServer = elaborateVectorFriends(v) {Obtains the list of friends of the chosen username present
in the server}
5: Vector bcServer = elaborateVectorBC(v) {Obtains the business card of the chosen username present in the
server}
6: for i = 0 to friendsServer.size() do
7:   if friendsServer.elementAt(i) is not present in friendsDevice then
8:     Inserts friendsServer.elementAt(i) into the local memory of the device
9:   end if
10: end for
11: if !bcDevice.equal(bcServer) then
12:   bcDevice = bcServer
13: end if
14: Shows an information alert

```

First the application gets the list of friends and the business card present in the local memory of the device (lines 1-2). Next it connects to the server using an internet access and it delivers as a parameter the list of friends present in the local memory of the device (line 3). Then the application receives the response of the server in the form of vector *v* and next it splits *v* into two separate vectors: friendsServer and bcServer (lines 4-5). The former vector contains the list of friends present in the server. This list is obtained by invoking the *elaborateVectorFriends* method which takes as an input parameter the vector *v*. The latter vector contains the business card present in the server, which is obtained by invoking the *elaborateVectorBC* method. This method also takes as input parameter the vector *v*.

Next, the application compares the list of friends obtained from the server and the list of friends present in the local memory of the device. If some element is missing in one of them, this element will be inserted in the list where it is missing (lines 6-10).

The last part of the process is to compare the business card obtained from the server and the one present in the local memory of the device. If they do not match, the application will update and synchronize the business card stored in the local memory with the one obtained

from the server (lines 11-12).

Finally, the application shows an information alert which states that the action has been successfully performed (line 13). After this action the list of friends and the business card placed in the local memory of the device and the one present on the server are the same.

Example code 18: How does the server deal with requests of clients? In the pseudo code given above only one connection is established between client and server. This connection is referred to in the 3rd line, where the client requests the server for an update of his data including his business card and the list of his friends.

The server takes as an input the username of the user and the list of friends stored in the local memory of the device. Next it elaborates on this list and it executes insert queries on the BATABLFriends whenever some friendship relation is present in the local memory but it is not included in the database. Afterwards, the server sends back to the client two parameters. The first one is the list of friends obtained using a Select query on the BATABLFriends table and the second one is the business card of the user obtained in the same way as in the *Login as another user* method.

All the actions that were described above may also be performed using the Web site of the application. A short description of the web site is given in Appendix A

My virtual world

This is the main action that the user can perform. To perform this action an internet access is not needed, because it uses the Bluetooth technology.

From the Figure 5.38-a, the user can click on the My virtual world command, and after that the application informs that this action is going to use the Bluetooth technology (Figure 5.38-b).

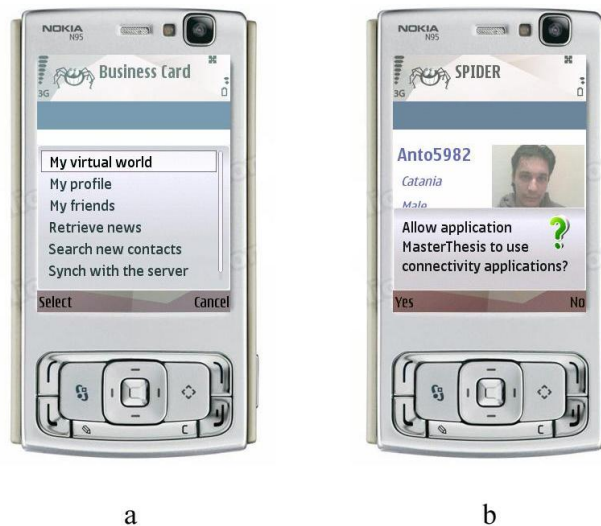


Figure 5.38: My virtual world action

The details concerning this action shall be explained in the next chapter.

A general idea of the implementation design of the Internet connectivity actions is presented in the form of the UML class diagram placed in Appendix E.

Chapter 6

My virtual world: Bluetooth connectivity

In the previous chapter the first part of the application regarding the communication between client and server was described. That part focused on the way users sign up to the application and create their own profiles, modify the profile, search for other users, check the list of friends, and finally, send messages and invitations. All these actions require an internet connection since the application has to communicate with the server to upload and download data.

The *My virtual world* action which constitutes the focus of the current section is based on the Bluetooth resource. The user can perform this action after having logged into the application. Both *Login* and *My virtual world* actions do not require an internet connection.

My virtual world aims at developing a framework where the Peer to Peer communication is controlled by a network of mobile devices. This network serves as an environment for exchanging various contents. Most of previously described actions using an internet access will be also available for the Bluetooth users. Specifically, users of the application will be able to discover other devices equipped with Spider and present within the Bluetooth range. They will be also able to discard Bluetooth devices which do not satisfy certain requirements. Once Spider users find each other, they can exchange their business cards and send messages and invitations. These are the most important actions available in the *My virtual world*, which together with less significant options will be explored in this section. A nice user interface resembling a game constitutes a strong part of the *My virtual world*.

At the beginning the way the user interface was implemented is going to be explained. Next, a profound explanation regarding the architecture of the *My virtual world* action will be given. Finally, all the possible actions that the user can perform using Bluetooth resource will be presented.

6.1 User Interface

The user interface provides a virtual reality, in which each user appearing within the Bluetooth range is represented by a virtual person.

The user interface was designed in the way which enables displaying different kinds of environment (Disco, Square, Bar, etc) without adding additional lines to the code.

In the following sections the design of the user interface and the way the user moves inside the user interface are going to be described.

6.1.1 Screen design

The screen design of the user interface is presented in Figure 6.1.

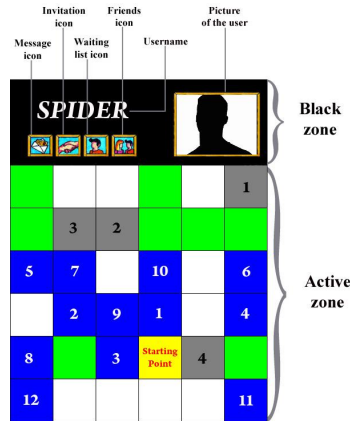


Figure 6.1: Screen design

The screen is divided into two main zones:

1. The black zone
2. The active zone

The black zone is positioned at the top of the Figure 1. It contains 4 small rectangles and one big rectangle. The first small rectangle constitutes an icon which informs if the user has received a Bluetooth message. The second icon is displayed on the screen when the user receives an invitation from another user. The third icon alerts the user that some people placed on the waiting list have entered the Bluetooth range. The last icon in turn informs the user that one or more of his friends have entered his Bluetooth range.

When the user approaches another virtual person on the screen, the picture of this person will be displayed in the big rectangle. Additionally, the username of the approached person will appear above the small rectangles. More details regarding these concepts will be given later in this chapter.

The active zone of the screen constitutes a big square divided into 32 small squares. These squares have different colors, which depict the role of each box. Below the description of the roles assigned to particular colors is given.

- The white boxes stand for the obstacles, for instance a table, a pool, soccer in, etc. The user which controls the person on the screen is not able to go through the white boxes
- The grey boxes signify places where the person on the screen can perform particular actions. Once the virtual person is located exactly in the grey boxes, the user can perform an action corresponding to the specific box by clicking on the fire button
- The blue boxes represent the places where other users entering in the Bluetooth range will be placed. Each box has a priority number. This means that if the application discovers three other users, these users will be placed in the three blue boxes with the lowest priority
- The yellow box is the initial place where the user is positioned after entering the virtual world
- The green boxes represent the places without any roles. It means that the user can go through these boxes and nothing will happen

Once that all the above mentioned rules are implemented, it is possible to create a user interface in an easy way, which does not require adding additional lines of code.

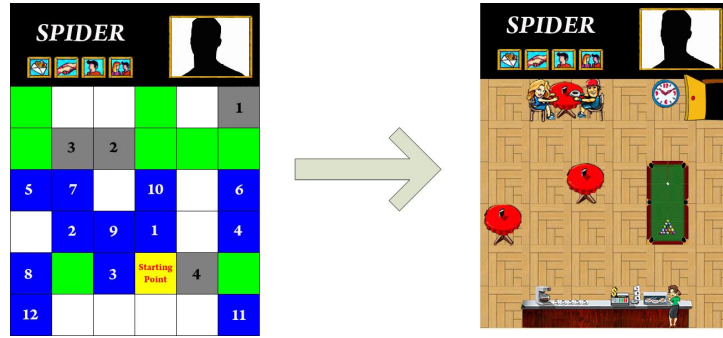


Figure 6.2: Application of the screen design

In Figure 6.2, it possible to see that the white boxes are replaced with tables, pool, and people. The grey boxes are not fulfilled with any objects, however if the user places the virtual person on these cells, he is able to perform particular actions assigned to these boxes. Apart from other users present in the Bluetooth range of the user, there are always three figurines displayed on the screen: a girl and a boy sitting at one of the tables and a barmaid. These people need to be approached by the user to enable performing a particular action. Specifically, the user needs to place the virtual person on the grey box which is close to the given figurine.

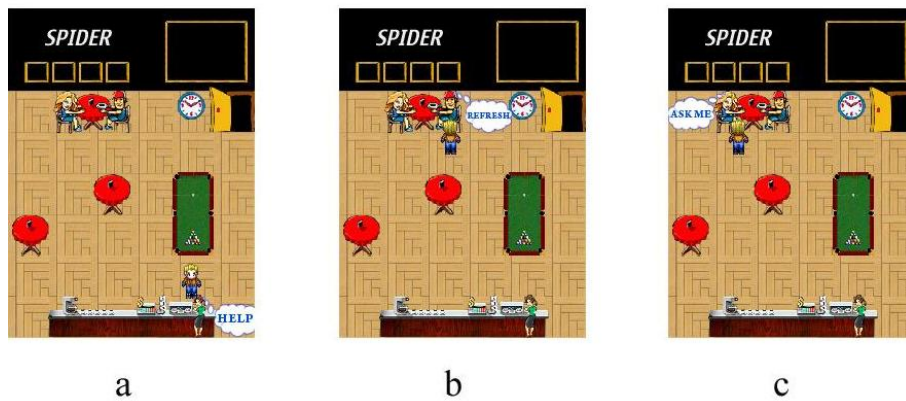


Figure 6.3: Possible actions

Figure 6.3 shows the three possible actions the user can perform:

- Help (Figure 6.3-a)
- Refresh (Figure 6.3-b)
- Ask me (Figure 6.3-c)

For instance, if the user wants to perform the Help action, he needs to get close to the barmaid and next, after the help text message appears on the screen, he needs to click on the fire button. The help text will appear when the user places the virtual person in the grey box close to the barmaid. For the Refresh and Ask me actions the process is the same.

The Help action provides the user with access to the help menu of the application. Under the Refresh action in turn, the user can refresh the screen by executing a Bluetooth discovery to check if new users possessing Spider applications entered his range and if the users previously found are still within his range. The last action available to the user is Ask me. When the user clicks on the Ask me command, the following actions can be performed.

- Search people: it is used to insert filters criteria, which enables to conduct a more specific search among all found people. For instance, a filter can help the user to find a desired username among the whole list of usernames present in his Bluetooth range.

- Chat: it is used to view sent and received messages
- Invitation list: it constitutes a collection of invitations received from other users
- Waiting list: it is selected if the user wants to be alerted each time when a specific friend placed on this list enters into his Bluetooth range

All these commands will be described in details in the section 6.3.

Another action that the user can perform is to approach the virtual people located within the Bluetooth range of the user. Figure 6.4 shows a scenario where virtual people are displayed in the screen of the username, which means that a Bluetooth discovery has already been conducted.

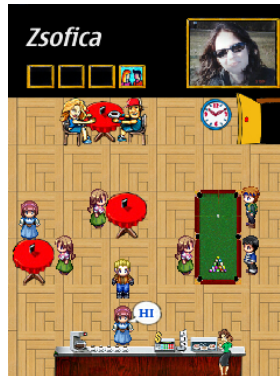


Figure 6.4: Approach a virtual person

When the user gets close to one of these virtual people, in the black zone of the screen the username of the virtual person will appear in the place where usually the Spider text is placed, and the picture of the user will be displayed in the big rectangle. This way the user is able to identify the people around him. Figure 6.4 presents the case when the user approaches a figurine and the username as well as the picture of the corresponding user are displayed in the black zone of the screen. Moreover, to better understand which figurine the user is approaching, close to this figurine a text message will appear saying "HI".

This is the way the user interface has been designed. What is worth stressing, thanks to such design of the interface changing the environment is not difficult.

To give an example, if the application provided an alternative seaside environment, the developer would need to build an image where in the place of the white boxes beach volley field, deck chair, etc. could be inserted. Close to the grey boxes, the developer could place people who lay down on these deck chairs. The user could perform the above mentioned actions after approaching these figurines.

Hence, this method of implementation allows for creation of multiple environments, among which the user could choose the one that he likes the most. This report treats this option as a potential future work.

6.1.2 Virtual people and timers

This section shall describe the way the user moves on the screen. Once the user selects the *My virtual world* action, the application displays a screen seen in the Figure 6.5-a. The user moves from the door to his initial position (Figure 10-a) thanks to a timer which is activated when the user interface is shown on the screen.

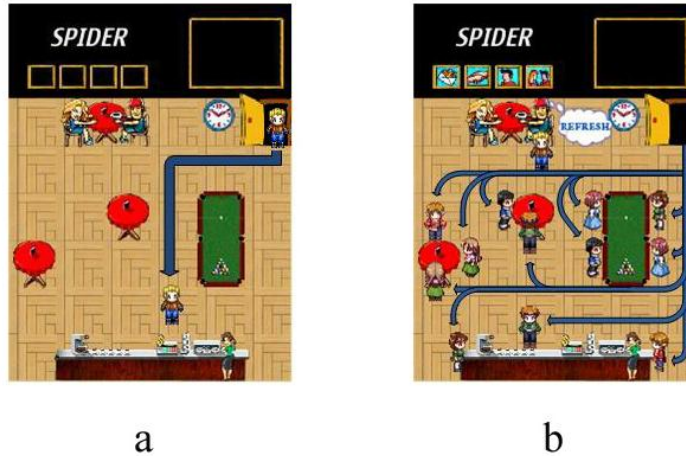


Figure 6.5: Virtual people paths

To be more specific, at first the virtual person is placed in the grey box on the top and on the right side in the Figure 6.1, which corresponds to the door in Figure 6.5-a. Then the virtual person goes to its initial position following a path which is implemented into the application. The application uses a timer to let the virtual person get the initial position. This timer is running till the user finds the right place on the screen.

Whenever the user discovers new users in his surrounding, these users will first appear in the upper right corner (the grey box) in the screen shown in Figure 6.1. The way they get their final positions marked by the blue boxes is the same as already described. This means that a path is implemented for each blue box and the timer is used for all users (Figure 6.5-b). Even if the displayed landscape will change, the path to each position of the virtual people is already stored and it will not change. The timer stops when all users get their final positions.

There are two possible states of the timer:

1. Timer activated: the user cannot move the virtual person since the application itself is moving the figurine
2. Timer deactivated: the user can move the virtual person

When the timer is deactivated, the user can move the virtual person on the screen by using the joystick of the mobile phone.

In order to move a virtual person in the screen, Java Micro Edition uses an important object called Sprite. A Sprite is an element that can be created with one of more frames stored in a single Image object. These frames are used to create animations displayed on the screen of the mobile phone. The Image used for a specific sprite is divided into equally-sized frames of a specified width and height. Figure 6.6 presents an image containing several frames [18].

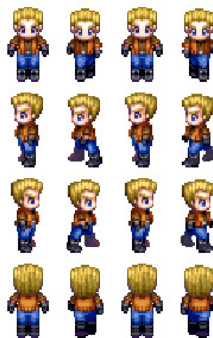


Figure 6.6: Sprite picture

The developer can divide the Figure 6.6 in 16 frames and then use them to generate animations. If the user moves the virtual person down the screen, the application will use only the first line of the Figure 6.6. The current pixel position of the virtual person on the screen will be refreshed according to the pattern:

$$(currentX, currentY) = (currentX, CurrentY + somePixels)$$

In this way the animation of the Sprite object moving down the screen is generated.

If the user would like to move the virtual person to the left of the screen, the application will only use the second line of the Figure 6.6. In this case the current pixel position of the virtual person on the screen will be refreshed according to the pattern:

$$(currentX, currentY) = (currentX - somePixels, CurrentY)$$

In this way the animation of the Sprite object moving to the left of the screen is generated. Similar logic is also applied to move the Sprite object up and to the right of the screen.

6.2 P2P application

As it was mentioned before, a Peer to Peer network is established when the *My virtual world* action is activated. Each node of the P2P network can simultaneously play two different roles: server and client. In fact, as it was shown in Figure 1.2, in each mobile device a server and a client are running in the same time.

The task of the server is to publish a service, whereas the task of the client is to search and connect to services. First a client conducts a device discovery, and then, for each discovered device, it conducts a service discovery. This way the application displays only the devices which publish the researched service, and discards the others. Once a client finds the desired service running in a server, it can establish a connection with that server.

Below a specific description of the server and the client is given.

6.2.1 Server

A crucial action performed by a server is to register a service. When a Bluetooth device connects to another device, it actually accesses a service offered by the other device.

Figure 6.7 presents the server's steps.

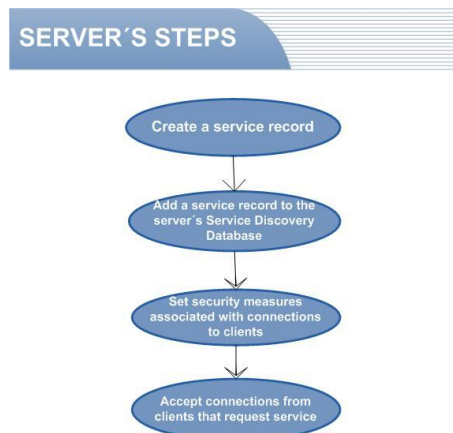


Figure 6.7: Server's steps

The first step is to create a service record. A service record contains a list of Universally Unique Identifiers(UUIDs) used to identify a unique Bluetooth service. Thanks to service record clients can connect to the desired Bluetooth service being offered by the server. As the second

step, the service record needs to be added to the Service Discovery Database. The Service Discovery Database keeps service records corresponding to the services offered by the devices. It means that server is able to offer one or more different services [15].

Then the security measures need to be set. In the Spider case, they are set to false. Since Spider application allows users to exchange only messages and invitations, setting the Authenticate and Authorization to the true value was not adequate for this application. In fact, in Spider application some actions are performed in background. Therefore, if the Authentication or Authorization was set to a true value, a PIN code would need to be inserted to send or receive data, and hence Spider would not be able to work in background. Setting the Authentication to a false value entails the necessity to set the Encryption to false value as well. This is because only when Authentication is set to true, the programmer may decide whether to set the Encryption to true or false. Even if all these values are set to false, it is important to stress that the server can receive messages exclusively from devices which are equipped with the Spider service.

The last step indicates that the server is ready to run and receive connections from the clients. Figure 6.8 presents the behaviour of the Spider server running in the mobile phones.

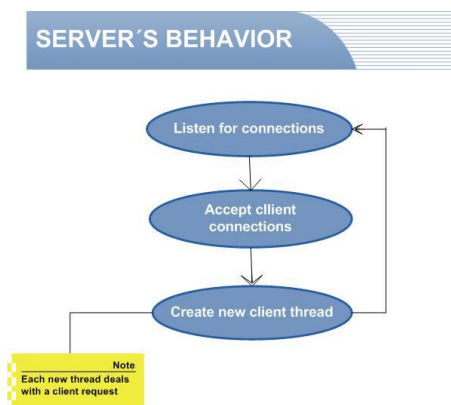


Figure 6.8: Server's behavior

In Figure 6.8 it is seen that the server is waiting for clients' connections. Once the server has accepted a connection, it will run a new thread which deals with the requests of the client. Concurrently, the old thread will continue to run waiting for requests of other clients. This way the server does not accept the connections sequentially, but can serve them simultaneously (it runs one thread per each client connection).

In section 6.3, a detailed description of all possible requests of the clients and the server's responses to these requests will be given.

6.2.2 Client

As already discussed, when a server is running, it publishes services. The client's task in turn is to locate and to access the services published by a server.

This section is devoted to detailed explanation of the way clients locate and access the services.

Figure 6.9 presents particular steps the client needs to take to perform these actions.

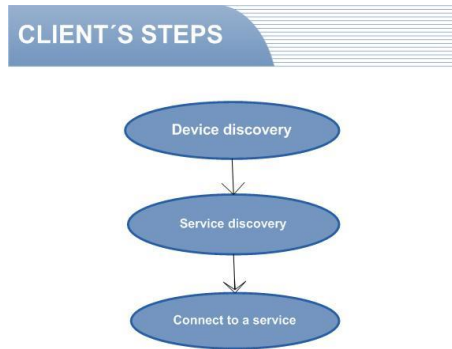


Figure 6.9: Client's steps

The three steps of the client to locate and connect to services are discussed below.

Device Discovery

The first step that the client has to follow is the device discovery, which consists in searching of all Bluetooth devices present in the range.

In the Bluetooth Specification there are two different inquiry modes: Limited Inquiry Access Code (LIAC) and Generic Inquiry Access Code (GIAC). The latter one indicates an unlimited search returning all devices, both in GIAC and LIAC mode, found in the Bluetooth range. The former one returns only devices which are in LIAC mode. For this reason the LIAC mode is faster than the GIAC mode. Unfortunately, the Nokia N95 which is the mobile phone used to test the application does not support the LIAC mode. In fact, the Nokia N95 belongs to the S60 3rd Edition FP 1 whose Symbian Bluetooth stack does not support the operation. Therefore, Spider adopted the GIAC mode [24].

When an inquiry is launched, the application returns all found Bluetooth devices, including the ones which are not running the Spider application. Therefore, certain filters have been implemented to narrow down the research to devices satisfying particular conditions. In particular, due to the fact that Spider is running only on mobile phones, the research does not embrace other sorts of Bluetooth devices. Including other kind of devices in the research would be tantamount to a waste of time and energy, since the found devices need to be analyzed afterwards. Fortunately, the optional package regarding the Bluetooth specification provides the opportunity to set some filters for the search of Bluetooth devices.

One of these filters is the major devices class. This filter groups devices into different classes. Table 6.1 shows some examples of major devices classes.

| Major device class | Example | Value |
|--------------------|-----------------------------------|-------|
| Computer | Desktop, laptop, PDA | 0x100 |
| Phone | Cellular, cordless, modem | 0x200 |
| Audio/video | Headset, speaker, video display | 0x400 |
| Peripheral | Mouse, joystick, keyboard | 0x500 |
| Imaging | Printer, scanner, camera, display | 0x600 |

Table 6.1: Bluetooth Major classes

Since Spider is running only on mobile phones, the major device class will be set to 0x200 value as shown in the table above. This way other Bluetooth devices discovered through the inquiry, for instance computers, PDAs or printers, will be discarded. Only mobile phones will be accepted.

Figure 6.10 shows the process of the device discovery.

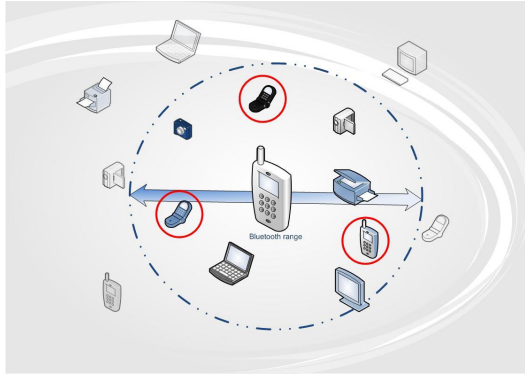


Figure 6.10: Device discovery behavior

In Figure 6.10 the device in the centre is conducting a search of Bluetooth devices. The mobile phones marked in red are filtered and all the other devices are discarded because they do not belong to the Phone major class.

However, it is unsure if the found Bluetooth devices are equipped with the Spider application. Therefore why Spider uses another filter to detect whether the *Spider_Service* is running on these devices. To apply this filter, a service discovery process is needed. This filter will be used on all devices which are part of the mobile phone major device class.

Service Discovery

Once the device discovery is completed, the found devices need to be analyzed by the service discovery. The service discovery searches the required service on each device. If some device satisfies this requirement as well, the two devices (the one which is conducting an inquiry and the one which is discovered) are able to communicate and exchange data. This mechanism is described below.

The service discovery takes as input the list of devices which are filtered by the device discovery using the appropriate major device class. Each device included in this list is inspected in terms of possessing the Spider service. To establish whether the desired service is running on that device, the method compares the uuid value and the name of the service that it is looking for with the ones offered by the inspected device.

Figure 6.11 shows the behaviour of the service discovery.



Figure 6.11: Service discovery behavior

The service discovery is investigating each mobile phone filtered from the device discovery. The mobile phone marked in red cannot connect with the phone in the centre of the picture since his server is not publishing the service called *Spider_Service*. The other two mobile phones in

turn can be connected because their servers are publishing the service which the mobile in the centre is looking for.

Connect to a service

Once some device satisfies the requirements set in the device discovery and service discovery processes, a connection can be established between this device and the one which conducted the Bluetooth research.

In order to connect with a device a *getConnectionURL* method is invoked.

```
public String getConnectionURL(int requiredSecurity, Boolean mustBeMaster)
```

This method allows for connecting with services. The first input parameter indicates the level of security for the connection. This level has to be chosen from the following options:

- No authenticate and no Encrypt
- Authenticate and no Encrypt
- Authenticate and Encrypt

In Spider application the security level is set on no authenticate and no Encrypt. The reason of this choice was explained in the section 6.2.1.

The second parameter of the method allows for being master or slave in the connection. This parameter may be set on true or false, where true value indicates that the device plays the role of master, while false value indicates that the device can be either a master or a slave.

Due to the fact that Spider has been tested on mobile phones running Symbian OS, *mustBeMaster* had to be set to false because the current implementation of Bluetooth API on Symbian OS supports only that value for the *mustBeMaster* parameter. If *mustBeMaster* is set to true, an exception will be obtained [15].

6.3 My virtual world actions

The previous section explored the way mobile phones communicate with each other through the Bluetooth technology. This section in turn shall describe possible actions available to the user under the *My virtual world*.

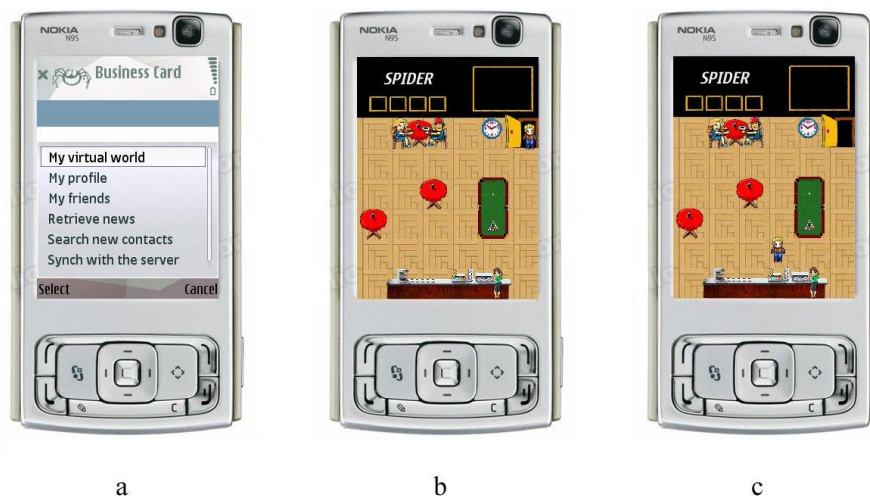


Figure 6.12: Start My virtual world

When the user clicks on the *My virtual world* command from the main menu shown in Figure 6.12-a, a server offering the *Spider_Service* starts to run - the device is set to be discoverable. Figure 6.12-b shows the user interface of the Spider application. Here, a virtual

person representing the user moves from the door to its initial position (Figure 6.12-c) as it was described in section 6.1.2.

When the virtual person reaches its initial position, the user is able to move the virtual person and perform several actions. The actions that *My virtual world* offers are divided in two parts:

1. Client actions: regard actions which can be performed by the user, i.e. start an inquiry, send a Bluetooth message, etc.
2. Server responses: regard the way the server deals with requests of the clients

In the following sections these actions are going to be presented in details.

6.3.1 Client actions

This subsection will describe in details the actions that the user can perform. A list of these actions is given below:

1. Help
2. Send Messages, Send invitations, Accept/Refuse invitations
3. Chat and User list
4. Search people
5. Setting waiting list
6. Refresh
7. Exit

All these actions are going to be described below.

Help

This action is performed by the user when he would like to consult the help menu.

The Help menu is useful for the user to understand the way the user interface works. It is rather intended for new users of the application, since in general the user will easily get the principal concepts of the *My virtual world* action simply by playing with the application, and thus will not need the Help anymore. A part of the Help screen is presented in Figure 6.13.

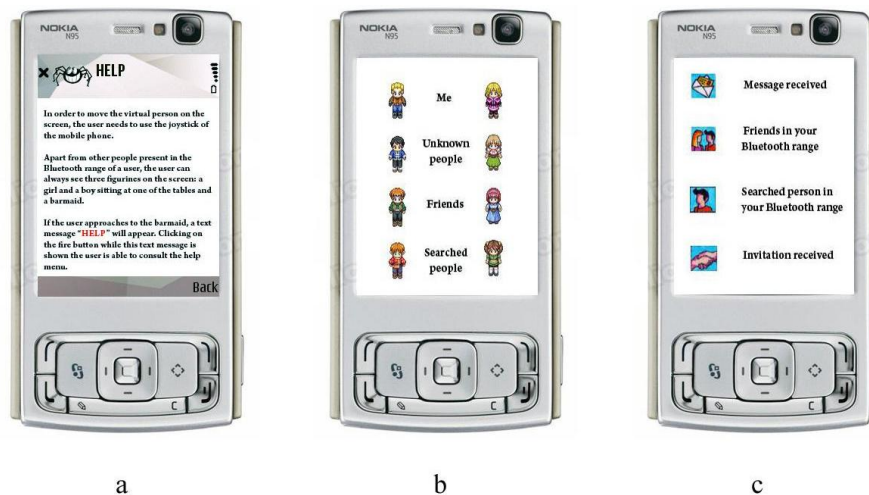


Figure 6.13: Help menu

Figure 6.13-b shows different kinds of figurines which can appear on the screen. On the left side there are figurines which appear when the user is male, while on the right side there are figurines representing female users. These figurines have different meanings: the virtual people in the first row represent the user of the application, figurines in the second row indicate unknown people, the third row regards friends of the user, and the fourth one represents people placed by the user on a waiting list. The waiting list is a list which can be set by the user to search specific friends. The user cannot place on the waiting list usernames which are not his friends.

Figure 6.13-c presents all icons that can appear on the screen of the mobile and their respective interpretation. The significance of these icons was already presented in section 6.1.1.

The pseudo code of the Help menu is presented below.

Example code 19 helpMethod(*Display* display, *Game* userInterface)

```

1: Image image = Image.createImage("help.png")
2: Canvas canvas = new Canvas()
3: canvas.drawImage(image)
4: Command back = new Command()
5: canvas.addCommand(back)
6: display.setCurrent(canvas)
7: if back is activated then
8:   display.set(userInterface)
9: end if

```

In the first line the image that the application shows in the help menu is stored in the *image* variable. Next, a canvas is initialized and the *image* is painted on it (lines 2-3). Then the canvas is displayed on the screen (line 6). If the user clicks on the back command, the user interface given as an input parameter will be shown on the screen (line 8).

Send messages, Send invitations, Accept/Refuse invitations

In this section the way the user can send messages and invitations to other users present inside his Bluetooth range is described. Additionally, it is shown how the user can accept and refuse an invitation received from another user.

Send messages In Figure 6.14-a, if the user clicks on the fire button while the username and the picture of *Zsofica* are displayed on the screen together with the text message saying "HI", the business card of the chosen username will be shown (Figure 6.14-b).



Figure 6.14: Send a message and Show profile actions

If the chosen username is a friend of the user, two actions are available: sending a Bluetooth message to the user and visualizing his or her full profile (Figure 6.14-c). Show Profile is the

only action which requires an internet connection. This action is conducted in a way which was described in the previous chapter. Send a Bluetooth message, instead, uses the Bluetooth technology. After the user clicks on Send Message command (Figure 6.14-c), the screenshot presented in Figure 6.15-a will appear.

In figure 6.15-a user is able to type the message he wants to sent to the chosen username. After having typed the message, the user needs to click on the Send message button. Then a confirmation message will be displayed (Figure 6.15-b).

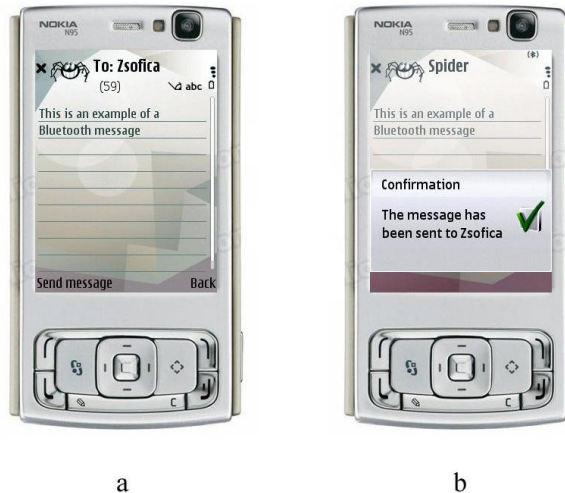


Figure 6.15: Type a message and get a confirmation alert

Send invitations This action is performed similarity to the send message action. After the user approaches an unknown virtual person (Figure 6.16-a), the business card of this person will be displayed (Figure 6.16-b). Then the user needs to select the *Send Invitation* command (Figure 6.16-c).



Figure 6.16: Send an invitation

After sending the invitation the application displays a confirmation alert.

Accept/Refuse Invitation Another two important actions available to the users are accepting and refusing invitations. When the user gets an invitation from another user, this invitation is displayed on the screen. Specifically, the application notifies the user about the fact that he received an invitation by displaying the corresponding icon (Figure 6.17-a).

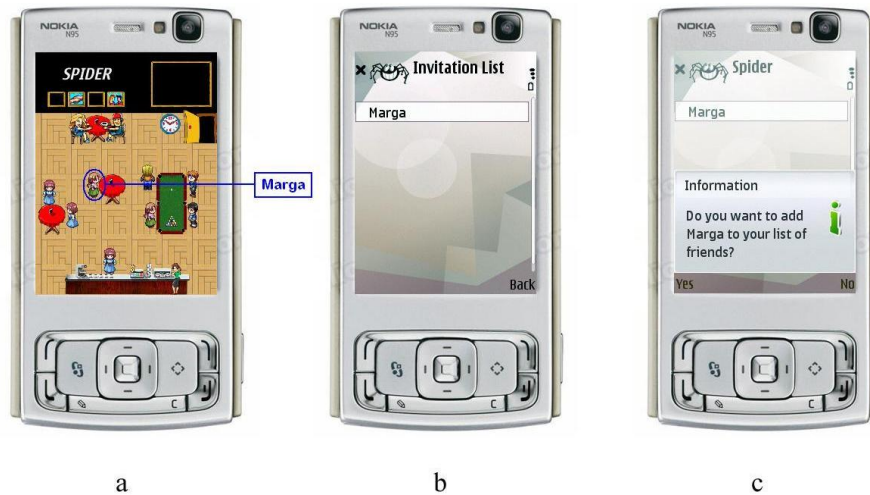


Figure 6.17: Accept or refuse an invitation

When the user wants to accept or refuse the received invitation, he needs to view the invitation list by approaching the girl seated at the table. This list includes all the invitations received so far (Figure 6.17-b).

In this example the invitation was sent by the username *Marga*. If the user clicks on this element of the list, the application will ask the user if he wants to accept or refuse the selected invitation (Figure 6.17-c). Regardless of the response, the invitation will be deleted from the list. In case the user accepts the invitation, the figurine which represents *Marga* will change from one representing the unknown person (Figure 6.17-a) to the one representing a friend status (Figure 6.18-b).



Figure 6.18: Result of accepting invitation

Additionally, a confirmation alert will appear saying that the invitation has been accepted (Figure 6.18-a).

Send messages and invitations issue Above the way the user sends invitations and messages to other users within his Bluetooth range was presented. When a user sends a message or an invitation, two possible scenarios can appear. The first scenario is presented in Figure 6.19.

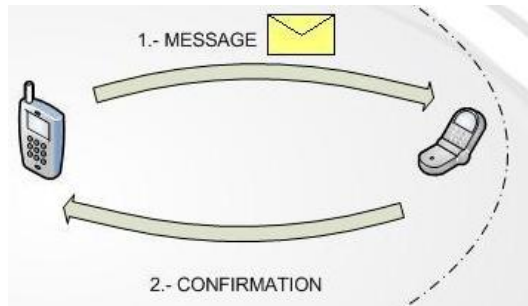


Figure 6.19: Send message: successful

In this scenario the user manages to send a Bluetooth message. The sender of the message knows that the message has been sent because he receives an automatic Bluetooth confirmation.

The second possible scenario is presented in Figure 6.20.

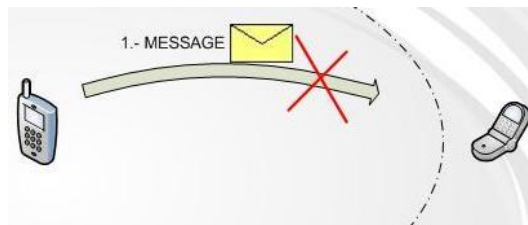


Figure 6.20: Send message: failed

In this scenario, when the user A did the Bluetooth discovery, the user B was within his range. However, the user B may change his position and get out of the Bluetooth range of the user A. If the user A does not refresh the screen, the figurine representing the user B will be still displayed on the screen of the mobile phone. In case the user A would like to send a message to the user B, the message cannot be delivered. Consequently, the sender will not receive the Bluetooth confirmation message. In this case the application reacts in the way presented in Figure 6.21.



Figure 6.21: Result of Send message action: failed

In Figure 6.21-a , the user Anto5982 would like to send a message to *Karola83*. Thus he gets close to *Karola83* figurine, and he follows the ordinary process of sending a message which was described before. Due to the fact that *Karola83* gets out of his Bluetooth range, after sending the message the application will display an information shown in Figure 6.21-b. After showing

the alert, the application will delete the figurine of *Karola83* from the screen (Figure 6.21-c). In case an error occurs during sending or refusing invitations, the behaviour of the application is the same.

However, the behaviour of the application changes when the user sends an accepted invitation message to the sender of the invitation who is not in his Bluetooth range anymore. In this case the user will receive another alert which is presented in Figure 6.22.



Figure 6.22: Result of sending the accepted invitation message: failed

When the user would like to perform one of these actions, the application invokes the *sendData* method shown in Example code 20.

Example code 20 *sendData(String data, Boolean isAcceptedInvitation, String url)*

```

1: boolean messageSent = send(url,data)
2: if messageSent then
3:   chat.append(data)
4:   Shows confirmation alert
5: else
6:   Shows information alert
7:   Deletes figurine of the chosen username
8:   Deletes the username form the list
9:   if isAcceptedInvitation then
10:    Stores the accepting invitation message in the local memory of the device
11:   end if
12: end if

```

This method takes as input the message which needs to be sent, the kind of the message, and the url which constitutes the address where the message needs to be sent. This method returns a boolean value depending on the fact whether the action of sending the message was successful (line 1). If the message was successfully sent, the application will shown on the chat form the text of the message (lines 2-3). Afterwards a confirmation alert is shown on the screen which informs the user that the message has been sent (line 4). In case the receiver is not in the Bluetooth range of the sender anymore, the application informs the user that it was not possible to send the Bluetooth message (line 6) and the figurine which represents the receiver is removed from the screen (line 7). The place on the screen that the receiver was occupying now will be set to free, so that it can be taken by other users entering the virtual world. Additionally, the receiver will be removed from the list of users present within the range (line 8). If the message constitutes the accepted invitation message, this message will be stored on the local memory of the device (lines 9-11). When the receiver of the message will appear within the Bluetooth range of the user, the message will be sent automatically and the friendship relation between the two users will be established.

Chat and Users list

This action is performed whenever the user wants to view the received messages. Typically, this action is used after receiving a message from other users. The user needs to click on the *Chat* command after approaching the girl seated at the table (Figures 6.23-a and 6.23-b). As a result the screenshot shown in Figure 6.23-c will appear. Here all the messages and invitations that the user has sent or received are displayed.



Figure 6.23: Chat action

If the user receives a message from another user, the application gives the opportunity to read the message from the chat form and reply to it without coming back to the user interface and searching the appropriate user among all figurines on the screen. This method enables to view all users present within the Bluetooth range in the form of list of their usernames. From the Chat screen, it is possible to click on the List Users command, and as a result the screenshot presented in Figure 6.24-a will appear. Figure 6.24-a shows a list of users present in the virtual world of the user.



Figure 6.24: User list action

When the user clicks on one element of the list of users, the business card of the chosen username will be displayed (Figures 6.24-b and 6.24-c).

This list of users is very important due to two reasons:

1. The user does not need to lose time to check each figurine in order to find the user who should receive a message or an invitation

- The application supports only 12 positions on the screen, which means that if there are more than 12 Spider applications within the range of the user, the application will show on the screen only 12 of them. However, all users will be displayed in the list of users.

Below the pseudo code of the chat action is presented.

Example code 21 Chat

```

1: if chat is activated then
2:   Adds list users command to chat Form
3:   Shows chat form
4:   if list users is activated then
5:     Shows the users list
6:     if an element of the list is selected then
7:       Shows business Card of the chosen username
8:       adds Send a message command
9:       if chosen element is a friend then
10:        adds Show Profile command
11:      else
12:        adds Send Invitation Command
13:      end if
14:    end if
15:    if Send message or send Invitation is activated then
16:      sendData(message type,false,service of the chosen element)
17:    else if Send Invitation is activated then
18:      sendData(invitation,false,service of the chosen element)
19:    else if Show Profile is activated then
20:      Shows the profile of the user (url)
21:    end if
22:  end if
23: end if

```

When the user selects the chat command the chat form is shown (line 3). From this form the user is able to click on the list users command (line 4). When the user clicks on this command, the list of all users present within the Bluetooth range is shown (line 5). Then, when the user clicks on a selected username, the application checks if the selected element is a friend of the user. If so, the user will be able to perform the *Send Message* and *Show Profile* actions; otherwise the *Send Message* and *Send Invitation* actions will be available (lines 6-14).

Search People

This action is performed by the user when he would like to conduct a search among users present in his Bluetooth range with a use of certain filters criteria. To have this option at the disposal, the user has to approach the girl seated at the table and click on the *Search people* command. The application will display the search form presented in Figure 6.25. This form is the same as the one presented in section 5.2.2. The difference between the two forms is the following.



Figure 6.25: Search people action

The values of Home City, Sex and Country fields, which can be selected to apply the filter criteria, are not all the distinct values placed in the database. Instead, they are only the distinct values characterizing people currently present in his virtual world.

Figure 6.26 shows an example of these fields.



Figure 6.26: Fulfillment of the Search people form

Figure 6.26, shows an example of a search among people present in the virtual world and possible values of fields which can be applied as filter criteria. Since these values can be combined in different ways, applying concurrently a filter on more than one value, ultimately it can turn out that no user from the *My virtual world* satisfies all criteria of the search.

Figure 6.27 shows an example of research with the use of filters criteria. In this case the user is looking only for *Polish girls* which are present in his Bluetooth range.

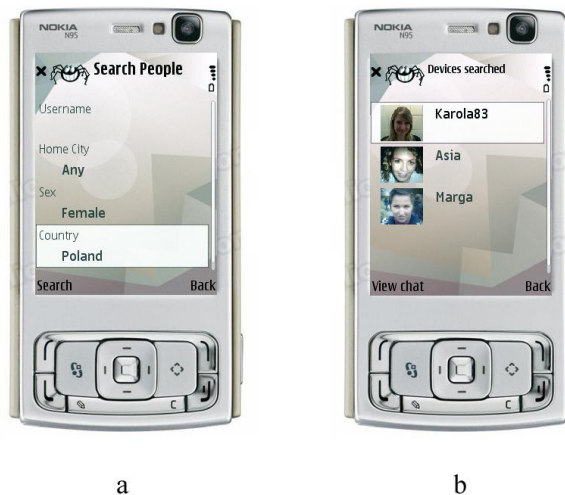


Figure 6.27: Example of the search results with filter criteria applied

After making the selections in Figure 6.27-a, the user needs to click on the *Search* command. Then the results will be presented in a list form (Figure 6.27-b). The user can select an element of the list and as a result, the business card of the chosen element will be displayed.

The pseudo code of the Search people action is shown below.

Example code 22 Search people

```
1: if search people is activated then
2:   ChoiceGroup sex = Any, Male, Female
3:   ChoiceGroup Home City = Any, all distinct Home city values of the users present in the virtual world
4:   ChoiceGroup || Country = Any, all distinct Country values of the users present in the virtual world
5: end if
6: {The user can type the pseudo username and make selections on sex, country and city fields}
7: List foundPeople = Shows found people
8: if Clicks in one of the elements of foundPeople then
9:   Shows business card of the chosen element
10:  adds Send Message command
11:  if chosen element is a friend then
12:    Adds Shows Profile command
13:  else
14:    Adds Send Invitation Command
15:  end if
16: end if
```

In the first four lines it is shown that the *ChoiceGroups* of Home City, Sex and Country are initialized. After the user makes his selections, a list of users satisfying these requirements will be displayed (line 7). If the user clicks on one element of this list, the business card of the chosen username will be presented and the user will obtain the possibility to send Bluetooth messages (lines 9-10).

Waiting list

This action can be performed if the user wants to be alerted when one or more friends will appear in his Bluetooth range. After clicking on the Waiting list option, the screenshot of Figure 6.27-a will appear. A list of friends of the user *Anto5982* will be displayed. In this example the user chooses two usernames from this list:

1. Zsofica
2. Nick

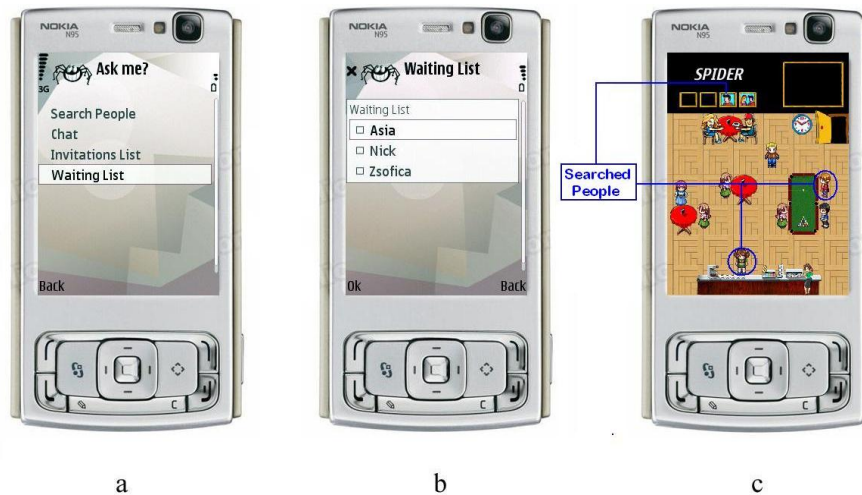


Figure 6.28: Waiting list action

After setting the waiting list the user will receive a confirmation. In case the two users placed on the waiting list are already present in the virtual world, in the user interface the two figurines will be changed from the ones representing a friend status to the ones signifying searched people. Moreover, the icon of the waiting list will be activated. In case these users are currently not present in the virtual world, the icon will be not activated. It will become activated only when the two people will enter the virtual world.

The pseudo code of the Waiting list action is presented below.

Example code 23 Waiting list

```
1: ChoiceGroup watingList = friends of the user
2: display.setCurrent(waitingList)
3: Vector waiting = User has to make his selections
4: {Saves Selections}
5: for i = 0 to deviceFound.size() do
6:   for i = 0 to waiting.size() do
7:     if deviceFound.elemtAt(i).equals(wating.elementAt(j)) then
8:       deviceFound.setFigurine(waitingStatus)
9:       ActiveWaitingIcon()
10:    end if
11:   end for
12: end for
```

In the first line the application retrieves the list of friends of the user from the local memory of the device. After that this list is displayed on the screen (line 2), and the user has the possibility to select some friends (line 3). If some of the selected friends are already in his Bluetooth range, the figurine representing the selected friend will change from the one marking the friend status to the one marking the waiting status. Additionally, the waiting icon will be activated (lines 5-12).

Refresh

The refresh action is the most important action under the *My virtual world*. This action is used to start an inquiry to discover Spider applications present within the user's Bluetooth range.

If the user wants to refresh the screen, he needs to approach the boy seated at the table. When a text message *Refresh* appears, he needs to click on the fire button (Figure 6.29-a). The application will start the Bluetooth discovery (Figure 6.29-b). Whenever the application finds some Spider application within his Bluetooth range, the two users will exchange their business cards.

After a while from selecting the *Refresh* command, the screen is refreshed and displays the found users as virtual people on the screen (Figure 6.29-c).

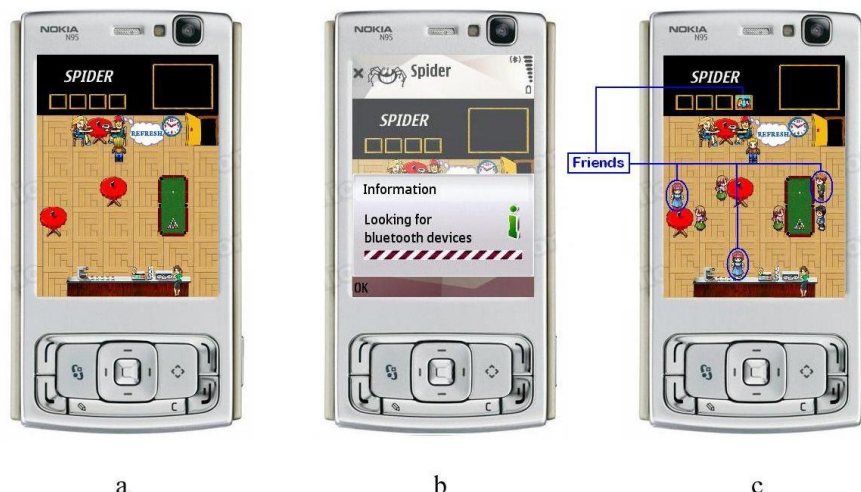


Figure 6.29: Refresh action

As we can see from the screenshot of Figure 6.29-c, the icon which signifies the presence of friends is activated. The icon means that one or more friends are present within the Bluetooth range of the user. If the screenshot from Figure 6.29-c is analysed, it can be noticed that there are three figurines which represent the status of a friend. Other figurines represent the status

of unknown people.

Below the pseudo code of the Refresh action is presented.

Example code 24 Refresh

```
1: Blocks the virtual person
2: Deactivate waiting and friends icons
3: Vector mobilePhones = Start device discovery()
4: Vector spiderApplicationsFound = Start Service discovery(mobilePhones)
5: Vector businessCard = new Vector()
6: for i = 0 to spiderApplicationsFound.size() do
7:   businessCard.addElement(getBusinessCard(spiderApplicationsFound[i]))
8: end for
9: if spiderApplicationFound.size()==0 then
10:   Remove all the figurines from the screen
11:   Delete all the elements from the users list
12:   Unblock virtual person
13: else
14:   listOfUsers.update(spiderApplicationsFound)
15:   updateScreen(spiderApplicationsFound)
16: end if
```

When the user is performing the refresh action, he is not able to move his figurine (line 1) till this process ends. Later on, the application executes three essential operations (lines 3-8):

1. Device discovery - which filters only mobile phones
2. Service discovery - which filters only devices providing the *Spider_Service* service.
3. Exchange business cards - for each device found, the device which started the inquiry opens an input and output streams to send its url address and the business card of the user. In return, it receives the business card of the discovered device

If no spider application is found, the method deletes all the figurines from the screen (lines 9-10) since they were representing users who had been placed in the Bluetooth range before refreshing the screen and they were not found in the range anymore after refresh method was invoked. Moreover, elements of the list under the chat form need to be deleted (line 11). Otherwise, if some Spider applications are found, the list of users needs to be updated with new usernames which appeared in the Bluetooth range and the *updateScreen* method is invoked (lines 14-15).

The pseudo code of the *updateScreen* method is shown below.

Example code 25 updateScreen(Vector spiderApplicationsFound)

```
1: Vector waitingFriends = people from waiting list
2: Vector friends = friends of the user logged in
3: Vector oldUsernameAlive = define all the users which are currently in the Bluetooth range and were also in
  the range during the previous discovery
4: Vector oldUsernameNotAlive = define all the users which were in the range during the previous discovery,
  but are currently not anymore in the Bluetooth range
5: for i = 0 to oldUsernamesNotAlive.size() do
6:   removeFigurine(oldUsernameNotAlive.elementAt(i))
7: end for
8: Vector newSprites = moveSprites(oldUseranameAlive, spiderApplicationsFound, friends, waitingFriends)
9: Send oldConfirmationInvitation();
10: Timer timer = new Timer (newSprite);
11: timer.run();
12: display.setCurrent(userInterface)
13: if timer is deactivated then
14:   Unblock virtual person
15: end if
```

The method initializes two important parameters which are useful for continuation of the process. The first parameter is the *waitingFriends* vector which contains all usernames that have been set in the waiting list (line 1). The second one is the vector friends which contains

all friends of the user logged into the application (line 2).

Then, the application checks if users previously found in the Bluetooth range are still among users found through the new discovery and placed in the *spiderApplicationFound* vector. If they are present in this vector, they become inserted in the *oldUsernameAlive* vector (line 3). If not, it means that they are not in the Bluetooth range of the user anymore, so they become inserted in the *oldUsernameNotAlive* vector (line 4).

In lines 5-7, for each element of the vector *oldUsernameNotAlive* the *removeFigurine* method is invoked. This method removes the figurines representing the users which are not anymore in the Bluetooth range and sets places on the screen previously occupied by corresponding figurines to empty places.

In the next line, the application invokes the *moveSprites* method. This method takes the *oldUsernameAlive*, *spiderApplicationsFound*, *friends* and *waitingFriends* as inputs. Next, it compares the *oldUsernameAlive* with the *spiderApplicationFound* inputs to find new users in the Bluetooth range found through this inquiry and not discovered through previous inquiries. Then, for these usernames it finds empty places on the screen and checks whether each username should have a status of unknown, friend, or waiting figurine. After these operations all usernames are inserted into a vector called *moveSprite*.

The application checks if there are some accepted invitation message (the information sent to the sender of the invitation confirming that his invitation was accepted) to be sent to the discovered usernames (line 9). Next, it runs a Timer to let the new figurines move from the door to their positions on the screen. Once they take the corresponding positions, the virtual person who represents the user is able to move.

Exit

To go out from the *My virtual world*, the user needs to go close to the door on the top-right of the screen and click on the fire button (Figure 6.30-a). The application goes back to the business card of the user (Figure 6.30-b) and from there the user can perform other actions or shut down the application (Figure 6.30-c).



Figure 6.30: Exit action

The pseudo code of the Exit method is shown below.

Example code 26 Exit

```
1: stopGame()
2: server.stopServer()
3: display.setCurrent(businessCard)
```

When the user goes out of *My virtual world*, the *Spider_Service* service offered by the user's device will be not available anymore. As a result, the user is no more discoverable for other Spider applications (line 2). Moreover, the business card of the user will be displayed on the screen (line 3).

6.3.2 Server responses

This subsection will give a detailed description of actions that the server of the mobile phone performs as a response to clients' requests. Example code 27 shows how the server deals with the clients' requests and decides about the way to answer to these requests.

Example code 27 run()

```

1: DataStream in = conn.openDataInputStream()
2: int idRequest = in.readInt()
3: if idRequest == IDBusinessCard then
4:   exchangeBC()
5: else if idRequest == IDMsg then
6:   receiveMsg()
7: else if idRequest == IDInvitation then
8:   receiveInvitation()
9: else if idRequest == IDRefInvitation then
10:  refuseInvitation()
11: else if idRequest == IDAccInvitation then
12:  acceptInvitation()
13: end if

```

This method is the run method of the thread that is created by the server each time it receives a request. Depending on the request of the client, the server reacts in various ways by invoking different methods which are described below.

Exchange business card

When the server of the user's mobile phone receives this request, reacts as it is shown in the pseudo code presented below.

Example code 28 exchangeBC()

```

1: Vector businessCard = Receives Business card
2: String service = Receives the url address of the sender
3: Sends my business card to the client
4: if the sender is not in my virtual world then
5:   spiderApplicationsFound.addElement(sender)
6:   businessCard.addElement(businessCard)
7:   services.addElement(service)
8:   updateScreen(spiderApplicationFound)
9: end if

```

First, the device receives the business card and url address of the sender of the request (lines 1-2). Afterwards the business card of the user is sent to the sender (line 3). Next it is checked whether the sender of the request is still in the *My virtual world* (line 4). If not, the business card and the service of the sender are stored (lines 6-7) and the *updateScreen* method is invoked (line 8). This way both devices know about the existence of the other device.

Receive a message

The server can also get a request to display a message which was just received. Example code 29 shows the way the server responds to this request of the client.

Example code 29 receiveInvitation()

```

1: chat.append(message)
2: activateMessageIcon()
3: Manager.playTone(100,200,100)

```

When the server receives the message, it displays it on the chat screen (line 1). To alert the user that a message was received, the message icon is activated (line 2) and the mobile phone rings (line 3).

Receive an invitation

This is the case when some other user sends an invitation to the user. The pseudo code of this action is presented below.

Example code 30 receiveInvitation()

```
1: if the invitation has not received before then  
2:   Stores on the local memory the invitation  
3:   ActivateInvitationIcon()  
4:   Manager.playTone(100,200,100)  
5:   chat.append("An invitation received from" +sender);  
6: end if
```

In this method, the server gets an invitation from some other user. First, the server checks if the invitation was received previously (line 1). If this is a new invitation, the server stores it in the local memory of the device (line 2). Then it notifies the user by a sound alert (line 4) and by activating the invitation icon (line 3). Additionally, a message stating that the invitation is received will be displayed on the chat form (line 5).

Receiving a refused invitation message

This is the case when the user has sent an invitation to another user who then has refused it. The reaction of the server to the refusal of the invitation is presented below.

Example code 31 refuseInvitation()

```
1: ActivateMessageIcon()  
2: Manager.playTone(100,200,100)  
3: chat.appendChat(Sender+" has refused your invitation");
```

Similarly to the previous method, the server alerts the user by writing a message on the chat form, activating the message icon and making the sound alert.

Receiving an accepted invitation message

This is the case when the user has sent an invitation to another user who then has accepted it. The pseudo code of this action is shown below.

Example code 32 acceptInvitation()

```
1: Stores in the local memory the friendships between the receiver and the sender of the invitation  
2: ActivateIconMessage()  
3: Manager.playTone(100,200,100)  
4: chat.appendChat(friend+" has accepted your invitation");  
5: if friend.isInTheRangeBT() then  
6:   setFigurineFriend()  
7: end if
```

First, the server stores the new friendship on the local memory of the device (line 1). It writes on the chat form that a new friend is added (line 4). Additionally, the message icon is activated to alert the user (line 2). Moreover, if the new friend is in the Bluetooth range of the user and is displayed on the screen of his phone, the old figurine which represents the unknown person will be replaced with the figurine representing a friend (lines 5-6).

A general idea of the implementation design of the Bluetooth connectivity actions is presented in the form of the UML class diagram placed in Appendix E.

Part IV

Testing and Evaluation

Chapter 7

Test and evaluation project

In this section the performed tests of the application are described. This phase of the development of the project has the target of finding weaknesses of the application.

All performed tests could be divided into three blocks:

1. Unit tests
2. Usability test
3. General evaluation

The Unit tests refer to the tests performed in order to check whether the application behaves how it is expected. The usability tests refer to the level of difficulty of using the application. Further, the general evaluation refers to the overall assessment of the application, including its strong and weak points, as well as suggestions on potential improvements. Below these tests and their results shall be presented.

7.1 Unit tests

Two different strategies have been used to test the Spider application:

1. Manual testing
2. Testing tool

The testing tool used to test the Spider application is J2MEUnit. J2MEUnit is based on the original JUnit and it contains the unit testing framework for Java micro edition [25].

In this section the results of the Unit tests are presented. Table 7.1 shows the kinds of tests and the obtained results.

| Test | Result |
|------------------------------------|---------------|
| Initialization and mission actions | Passed |
| My virtual world actions | Passed |
| Bluetooth limitations | Failed |

Table 7.1: Bluetooth Classes

The first set of tests aimed at checking if the application was behaving exactly how it was expected when a user was performing the initialization and mission actions. The tests are described in Appendix B.

The second set of tests aimed at checking whether the application was behaving exactly how it was expected when a user was performing the main My virtual world actions. The details of

these tests are given in Appendix C.

The third part of tests regarded the Bluetooth limitations. These tests are described below.

Bluetooth limitations tests

Test case: Send a message during a Bluetooth inquiry

- Description of the test: It was tested whether the user A was able to send a message to the user B while the latter one was doing a Bluetooth inquiry
- Expected result: The message has to be received by the user B
- Actual result
 - The application alerts the user A that the receiver is not anymore in the Bluetooth range
 - The application deletes the figurine of the user B from my virtual world
 - The application deletes the username of the user B from the User list (the user list is available under the chat action)
- Result: **FAILED**

Motivation of the failure The reason of this failure is that when a Bluetooth device starts an inquiry, it is not discoverable for other Bluetooth devices. It means that when a device is doing an inquiry, it cannot be found by other devices even if in reality it is present in their Bluetooth range. Hence other devices cannot connect to the device which is at the same time doing an inquiry [26].

Possible solution When a user sends a message to another user who has already started an inquiry, the application could behave in the following way:

- The application will notify the sender that for the moment the message has not been sent
- The figurine and the username will be not deleted from the screen and from the users list respectively
- The message will be sent after 45 seconds, which is the average time of the Bluetooth inquiry

If after 45 seconds the user previously doing an inquiry will be still present within the sender's Bluetooth range, the message will be sent successfully, the sender will be alerted and the receiver will remain in the user's virtual world. If the message will not be sent after 45 seconds since the receiver will move out of the range, the receiver will be deleted from the virtual world of the user, and the sender will be notified.

This solution is already added to the current implementation of Spider.

Test case: Concurrent Bluetooth inquiries

- Description of the test: It was tested whether two devices starting a Bluetooth inquiry at the same time were able to discover each other
- Expected result: The devices have to find each other
- Actual result:
 - 0% Device A finds device B and B finds A
 - 75% Device A finds device B and B does not find A
 - 25% Device A does not find B and device B does not find A
- Result: **FAILED**

Motivation of this failure The reason of this failure is the same as in case of the previous test. In fact, in case of automatic Bluetooth discovery, it is recommended to make the intervals between inquiries random. In this way, if two inquiries collide one time, probably they will not collide the next times.

Possible solution The following proposed solution is not going to solve the problem completely, but it will diminish the percentage of cases of failure.

The possible solution is the following: each time a device discovers another device, both of them will know about the existence of the other device. For example, if the device A starts an inquiry and finds device B, they will exchange the business cards and the device A will also send its url address. Thanks to this address the device B can connect to the device A and send a message even if during its inquiry the device A was not discovered. This concept was described in section .

This solution was implemented in the Spider application. In this chapter the technique proposed as the solution to this Bluetooth pitfall is called Refresh 2.0.

Before this test was performed, the refresh method which Spider was using was implemented in a different way. In fact, in the previous implementation, when a user A discovered the user B, only the user A knew about the existence of the user B and the business card was only sent from the discovered to the discoverer and not in both directions. In order to know about the existence of the user A and obtain his business card, the user B needed to discover the user A through an inquiry. In this chapter this technique is called Refresh 1.0.

The adopted solution (Refresh 2.0 method) allowed to increase the percentage of successful message deliveries in comparison to the Refresh 1.0 method. The results and frequency of their occurrences are presented below:

- 75% Device A finds device B and B finds A
- 0% Device A finds B and B does not find A
- 25% Device A does not find B and B does not find A

To better understand and evaluate the proposed solution three more tests have been performed in a scenario where 6 mobile devices are running. In these tests both methods: the Refresh 1.0 and the Refresh 2.0 were used and the obtained results were compared.

Test case: On 6 Spider applications 2 start an inquiry at the same time

- Test description: In the environment 6 Spider applications are placed. Two of them start a Bluetooth inquiry at the same time
- Expected result:
 - The devices which start an inquiry should have five virtual people in their virtual world
 - The other devices should have two virtual people in their virtual world
- Actual result: results are shown below

Figure 7.1 presents the results of this test. In the axis X the devices are placed; the first two devices in the axis are the ones which start an inquiry at the same time. Axis Y presents the percentage of devices found by the six devices.

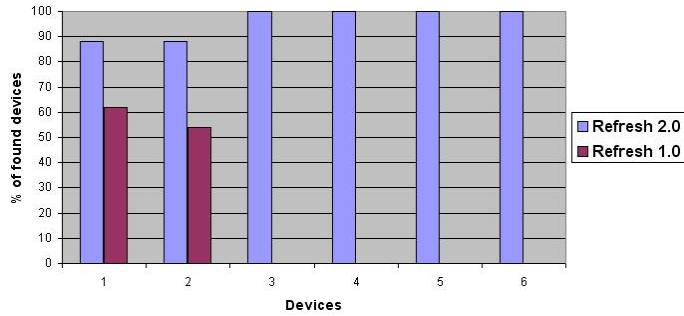


Figure 7.1: Spider discovery - scenario 1

This test shows that the Refresh 2 method does not allow for discovering 100% of devices, however in comparison to Refresh 1.0 method it improves the results of the search considerably. On 18 possible discoveries of devices (5 by each of the two devices doing an inquiry and 2 by each of the remaining 4 devices), the Refresh method 2.0 enabled to find 93,33% of the devices, and the Refresh method 1.0 enabled discovery of 32,22% of the devices.

Test case: On 6 Spider applications 4 start an inquiry at the same time

- Test description: In the environment 6 Spider applications are placed. Four of them start a Bluetooth inquiry at the same time
- Expected result:
 - The devices that start an inquiry should have five virtual people in their virtual world
 - The other devices should have two virtual people in their virtual world
- Actual result: results are shown below

Figure 7.2 presents the results of this test. In the axis X the devices are placed, among which the first four devices are the ones that start an inquiry at the same time.

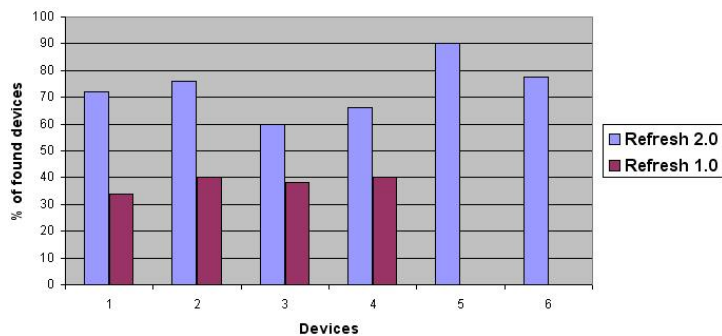


Figure 7.2: Spider discovery - scenario 2

In this case on 28 of possible discoveries, the Refresh method 2.0 allowed to find 72,85% of the devices and the Refresh 1.0 enabled finding 27,14% of devices. Therefore, this case also confirmed the advantage of the Refresh 2.0 over the Refresh 1.0 method.

Test case: On 6 Spider applications all of them start an inquiry at the same time

- Test description: In the environment 6 Spider applications are placed. All of them start a Bluetooth inquiry at the same time
- Expected result:
 - All of them should have five virtual people in their virtual world

- Actual result: results are shown below

Figure 7.3 presents the results of this test.

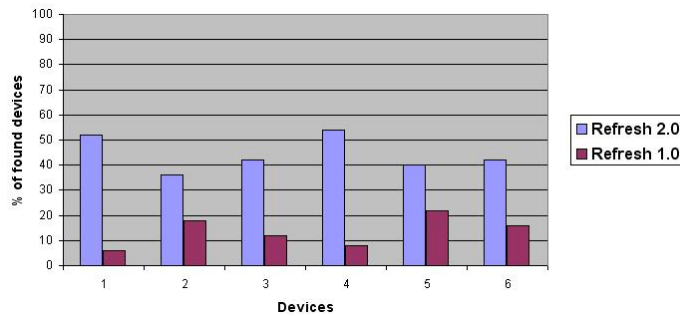


Figure 7.3: Spider discovery - scenario 3

The maximum number of possible device discoveries is 30. The Refresh 2.0 enables finding 44,33% of the devices and the Refresh 1.0 method allows to find 13,66% of the devices.

The aim of employing the Refresh 2.0 method was not to solve the problem of Bluetooth limitation, but to find a way to obtain the possibly maximum number of connections even in a scenario where this limitation still exists.

Another solution for this problem was considered and examined. Once a device A is discovered by a device B, the device A sends to device B not only its own business card, but also the business cards and the url addresses of all people present in his virtual world. However, this solution was discarded due to the possibility that the user A from this example may have done the recent Bluetooth discovery long time before and hence may send to the user B business cards of people who are no longer in the Bluetooth range. This would constitute a loss of time and that is why this solution was rejected.

7.2 Usability tests

The usability test was done by a group of students of the Aalborg University during the "Programmable digital units - Software test" classes.

The students were asked to evaluate the usability of the application. They were asked not to use the Spider user guide or the Spider F.A.Q. since typically, when customers get a new application on their mobile phone, they do not read the instruction of the game and they run it immediately. In cases when students testing the application were not able to perform a particular action, they were allowed to consult the Spider user guide or the Spider F.A.Q. These guides are placed in the web page of the project (Appendix D).

The students were asked to report bugs in the application in case they found any. Later on, the found bugs were corrected.

The questionnaire which was fulfilled by the students is presented in Appendix D. The performed usability tests are grouped in three main categories:

1. Initialization actions: students were asked to perform the initialization actions described in section 5.2.1
2. Mission actions: students were asked to perform the mission actions described in section 5.2.2
3. My virtual world actions: students were asked to perform the My virtual world actions described in section 6.3

The results of these tests are presented below.

7.2.1 Initialization actions tests

Figure 7.4 shows the result of this test.

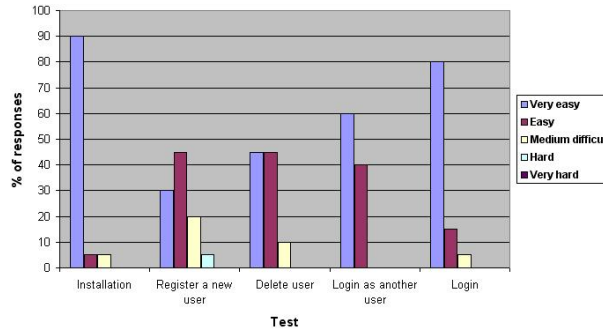


Figure 7.4: Initialization actions tests

The results of the Initialization actions tests show that almost all the actions were evaluated as "Very easy" to perform. Only the "Register a new user" action was mainly assessed as "Easy" to perform. The results of these tests suggest that the usability of this part of the application does not require major improvements. Perhaps in the next implementation of Spider, the Registration action could be analyzed once more and made easier for customers.

7.2.2 Mission actions tests

Figure 7.5 shows the result of this test.

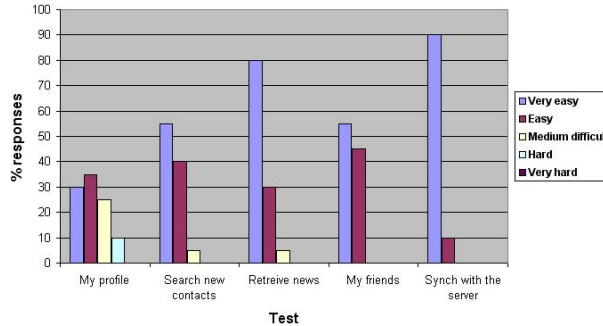


Figure 7.5: Mission actions tests

Also in the case of the Mission actions tests the results are good; in fact in almost all the tests the majority of responses was "Very easy". Only in "My profile" test, the majority of responses was "Easy" and there were relatively high per cent of "Medium difficult" responses, as well as few per cent of "Hard" responses. Probably, the design of this action needs to be reviewed in the next implementation of Spider to be easier for customers.

7.2.3 My virtual world actions tests

Figure 7.6 shows the result of this test.

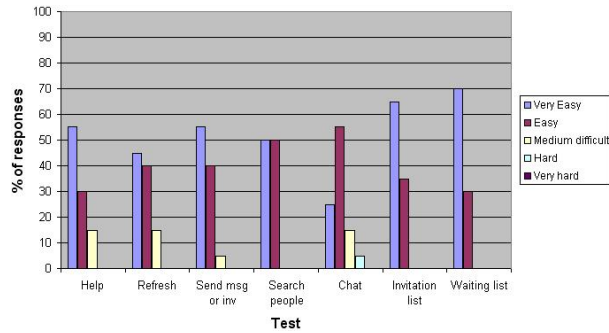


Figure 7.6: My virtual world actions tests

The results of the My virtual world tests show that these actions were "Very easy" to perform. Only the "Chat" action was assessed as relatively difficult; actually also few per cent of the "Hard" responses was obtained. Probably, in the next implementation of Spider the way the chat is visualized could be studied in order to make it easier for customers.

In general these results suggest that Spider is easy to use and customers are able to understand the functioning of the application quickly.

The obtained results are particularly interesting since the tests were performed by people who were not familiar with the application and have not read the application guide. Probably, reading the guide could help the customers to get to know the application in depth. Unfortunately, this is not the typical way customers behave.

7.3 General evaluation

At the end of the questionnaire some general questions were asked to the students. Some of them had a form of multiple choices; others were done as open questions. The responses on the questions which could be selected among multiple choices are presented as graphs. Later on, the most interesting answers to the open questions are reported.

The following questions were asked in the form of the multiple choice test:

1. Interaction with the user: How good was the interaction with the user?
2. Game flow: How good was the game flow of the application?
3. Spider: How do you evaluate the idea of Spider?
4. My virtual world: How do you evaluate the idea of My virtual world?
5. HTTP: How do you evaluate the HTTP actions regarding the fastness and robustness?

The results are presented in Figure 7.7.

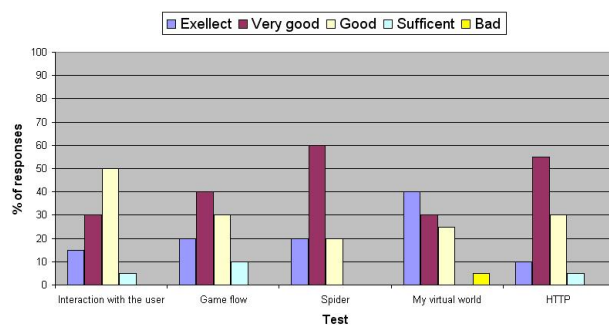


Figure 7.7: General evaluation

On the basis of these results it can be concluded that the Spider application was evaluated as an interesting application. In fact 75% of the students thought that it was an innovative idea in the sense that it was not widespread on mobile devices yet. The My virtual world part was evaluated particularly high; it received the highest per cent of "Excellent" responses. These tests also suggest that some improvements could be done in the application. In fact, the highest per cent of the responses regarding the Interaction with the user was "Good", and probably an easier design of the application could increase this value. More details are given in the part regarding the future work of the application.

As open questions, the students were asked the following:

- What they really liked in the application
- What they did not like
- What could be improved in the application

The responses of these questions are given below.

What they liked:

- Notification that friends and searched people enter the virtual world
- The idea of talking to other users
- Chat room
- The icons on the top of the screen
- Seeing the picture of the user when you get closer to the virtual person
- The idea of exchanging the business card
- Each user has own profile which is available only to the user's friends. This way it is easy to get info about friends
- User interface of the application

What they did not like:

- Some parameters of the profile are unnecessary (ex. Date of birth)
- The Bluetooth discovery was too slow
- Manual Bluetooth discovery
- Short range
- To refresh the screen you always need to approach the guy seated at the table.

What could be improved:

- HTTP - Timeout of the retrieve news: It was not implemented because this way the internet connection would need to be always opened or one needs to be opened for each timeout
- My virtual world - the HELP lady should be always highlighted or stop the user at the first time he gets close to her: this was a good observation because when a user launches the My virtual world for the first time, he doesn't know about the existence of the help menu. Thanks to this observation, this idea was introduced to the Spider application
- My virtual world - Bluetooth discovery should be done automatically: this idea was taken into consideration at the beginning of the implementation, but it was discarded because of the high energy consumption during the Bluetooth discovery

- My virtual world - Add fast commands to perform the "Ask me", "Refresh", "Help" actions (similarly as "CTRL C" and "CTRL V" shortcuts are used to copy and paste some text in the PC): this was a good observation, which is already introduced in the current version of Spider
- My virtual world - Place the people in the same range in relation with the distance of the Bluetooth devices from the user: this is not possible to be implemented
- My virtual world - Clean up the chat form. When the chat form is full of messages, a clean up command could be useful to clean the chat form: thanks to this observation this idea is already implemented in the current version of Spider
- My virtual world - Increase the speed of the Bluetooth discovery and the Bluetooth range

The last point of this list is really interesting. To better understand how slow the Bluetooth discovery is, Figure 7.8 presents a graph where on the axis Y there are the milliseconds passed till the Bluetooth discovery ends and on the axis X there is the number of found devices.

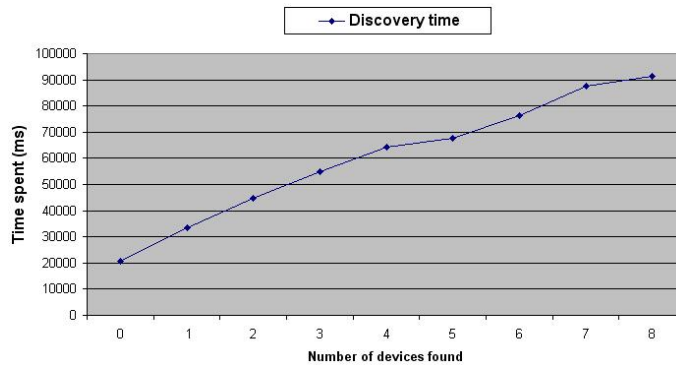


Figure 7.8: Bluetooth discovery time

As it is shown in Figure 7.8, the Bluetooth discovery is really slow. Unfortunately, there are no ways to make it faster. The only way to increase the usefulness of Spider regarding the range and the speed of the application is to modify the structure of the architecture of the project. This idea will be presented in the Future work section.

Part V

Conclusion and future work

Chapter 8

Conclusions and Future work

8.1 Conclusions

The main motivation to create Spider arose from the great success of various social networks available for Internet users. Their popularity is related to changing lifestyles, where people increasingly look for new acquaintances or cultivate old friendships exactly through the use of Internet, typically using their PCs. Yet along with the rapid growth of the number of mobile phone users, as well as with the development of so-called smart phones, the need for creating similar services for mobiles arises. What is important, the mobiles are not just entry points to existing social networks with their centralized architectures; being mobile offers one more degree of freedom.

Therefore, this project aimed at creating a social network application for mobile phones, providing the application users with an opportunity to make acquaintances, search people on the basis of certain criteria, chat, view profiles of other users, and with numerous other possibilities. Yet the application not only offers the mobile phone users similar service to social networks available in the web. Spider moves beyond this concept and gives also an opportunity to interact with other application users through the Bluetooth connectivity. Hence a user can get to know people situated in his proximity: their usernames, nationalities and pictures.

Initially, an architecture to implement this idea was submitted, and at the end it can be summarized that the general concept of the social network was successfully realized and objectives established at the beginning of the project were accomplished. However, the implementation of Spider turned out to be a challenging task involving development of extensive constituent parts as well as synchronization of their operation. In particular, it required analysis and evaluation of potential resources serving to build Spider, creating a database with advanced algorithms, applying servlet, using the Bluetooth technology, developing a web site, implementation of a user interface, and so on. Along with the development of the application, numerous challenges occurred, just to mention difficulties in linking and synchronizing various resources or problems stemming from the Bluetooth limitations.

At the end of the project it can be said that the application works smoothly. This was achieved thanks to numerous tests which were conducted both by the developer and by first users testing the application. When an error or a bug was found by a tester, a potential solution was searched, analysed and implemented. As a result, one may say that the current version of Spider already represents an advanced level of implementation leaving only some smaller issues to be solved and further tested. Therefore, Spider is not a commercial product yet; it still has some drawbacks and limitations discussed in the future work section. Moreover, more tests (especially in a large scale) would be needed to affirm that Spider is reliable.

The current version of Spider can be downloaded from the web page of the project. Spider will participate on the "Forum Nokia Mobile Innovation Competition 2008 for University Students".

8.2 Future Work

The current version of Spider is not the final product. Some extra improvements can be added in order to make Spider a very interesting commercial product. These possible improvements are described below.

An easier registration form design

During the testing phase a considerable number of students considered the current registration form to be complicated. An easier design of the registration form could be implemented in the future version of Spider.

Various environments for the user interface

As it was explained in section 6.1.1, introducing additional environments would be an easy task thanks to the design of the My virtual world user interface. In this way the user would be able to choose in which environment he would like to play.

Increase the security

For the kind of data that the current version of the application contains, specific security measures were not necessary. However, if the application is expanded so that the information regarding the users will contain also sensitive data, the security of the application would need to be increased.

Automatic removal of virtual people

Another possible improvement could be an automatic check whether previously found users are still present within the Bluetooth range of the user. This "search" does not start a Bluetooth discovery, since it would be disadvantageous for the application due to its time and energy consumption. To achieve this objective, the following technique could be used: a message stating "Are you alive?" could be sent to users present in the virtual world repeated in fixed time intervals. If a particular user is still within the Bluetooth range, an automatic Bluetooth reply will be received. In case the user will not receive the reply, the corresponding virtual person will be removed from the virtual world of the user.

Increase speed of the discovery and range of the Bluetooth

Results of conducted tests described in section 7.1 suggest that the speed and the range are the two important weaknesses of Spider. Unfortunately, these weaknesses do not depend on the implementation of the application but they stem from the Bluetooth limitations. Apparently there is no valid solution to these problems using the actual P2P architecture. The only way would be to wait for a new implementation of the Bluetooth or a new wireless technology which would not be laden with these limitations. Probably, changing the architecture of the P2P could lead to a possible solution for these weaknesses of the application. Figure 8.1 shows the idea which could solve the range and the speed problems.

Figure 8.1 shows a scenario where several Bluetooth access points are placed. The access points should be equipped with Bluetooth device class 1 (range of 100 meters) and have two crucial goals:

1. Find Spider applications in their zone
2. Deliver messages

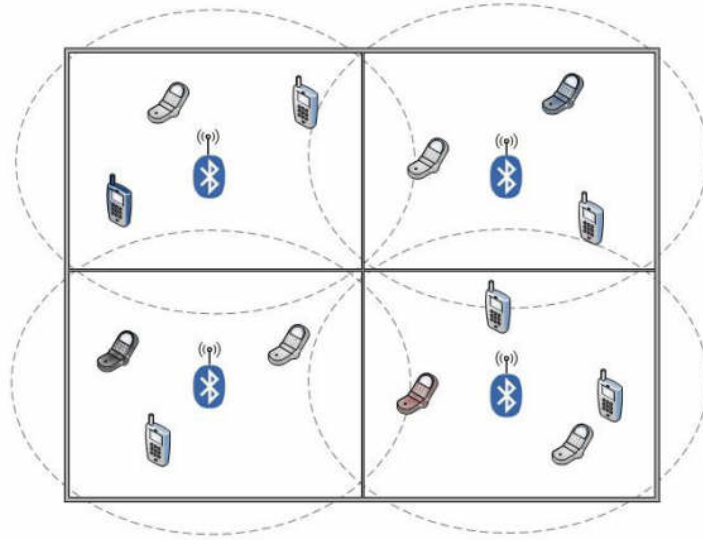


Figure 8.1: Future work P2P architecture proposal

The first target for each access point is to start an automatic Bluetooth inquiry repeated in fixed time intervals and share the list of people present in its range with all other access points. Whenever the Spider users would like to refresh their screens, they will not start a Bluetooth inquiry but they will connect with the closest access point requesting for the list of users present in the zone. The list of all people found in the area covered by access points will be returned.

The second target for each device is to deliver messages that the sender would like to send to some receiver. In fact, in this P2P architecture, when the sender would like to send a message to some other Spider users, the message will go through the access points.

This idea has important advantages:

- **Range:** using this implementation, the range of the Bluetooth device is not anymore 10 meters, but depends on the number of access points present in a particular area. For example, four Bluetooth access points could cover an area of 200 square meters (Figure 8.1). In this example a user will be able to discover Spider applications and sends Bluetooth messages within 200 square meters
- **Speed:** using this implementation, the Spider users will not have to do the Bluetooth discovery; the discovery will be done by access points. This way a lot of time to refresh the screen is saved
- **Localization:** using this implementation, the application gets an additional advantage: it is possible to localize people within the area covered by access points. As a result of the Bluetooth discovery, it will be possible to know in which zone of the whole area the people are present

This idea brings also an important disadvantage which is the fact that the operation of the Spider application would be restricted only to the specific area where Bluetooth access points are placed.

A possible solution could be to implement a Bluetooth discovery which works in the following way: when a user wants to refresh the screen, the application will try to connect to the Bluetooth access points. If none of the access points is available, the normal Bluetooth discovery will be conducted.

It would be nice to test this idea in a place where a lot of people work, for instance in a university campus.

Part VI
Appendix

Appendix A

Web site

The web site offers the same options to the user as the initialization and mission actions described in section 5.2.1 and 5.2.2. Due to the fact that the implementation of the web site was not a key target of this project, and since these actions were previously described, this section will not get into details of its implementation.

The web site provides an alternative way for the user to register into the application and perform the mission actions. Obviously, it regards only the HTTP actions and not the *My virtual world* actions. Figure A.1 shows the initial page of the web site.



Figure A.1: Iniatial web page

Here three possible actions can be performed:

1. Login, after inserting the username and password correctly
2. Delete user, which is possible after inserting the username and password correctly
3. Register a new user, which will open a registration form shown in Figure A.2

Spider - Registration

Username: anto5982
 Password: *****
 Re-write Password: *****
 Name: Antonio
 Surname: Sapuppo
 Home City: Catania
 Phone Number: +39095229855
 Email Address: anto@gmail.it
 Web Page: www.anto.it
 Date of birth: 5 / 9 / 1982

Sex: Male Female
 Country: Italy
 Languages: Danish, English, France, German, Italian, Spanish
 Interests: Music, Sport, Cars, Looking for girls, Looking for guys, Politics, Parties, Fashion

Buttons: Back, Register

Figure A.2: Registration form

The registration form presented in Figure A.2 is the same as the one displayed when a user registers using the mobile phone. After fulfilling the fields of the registration form, the user needs to click on the *Register* command. If all the parameters were inserted correctly and the chosen username was not already included in the database, the created profile is inserted in the database and the login is done automatically. After logging in the web site three tabs are available:

1. Data: Shows the profile of the user who is logged in
2. Friends: Shows friends of the user and the received messages and invitations
3. Search new people: Used to find people already registered in the database applying some filter criteria

These tabs are described below.


Data

If the user clicks on the *Data* tab, Figure A.3 will appear.

Spider - User Data

Navigation: Data | Friends | Search new people

Profile Information:

| | | |
|---------------|----------------|--|
| Username | Anto5982 |  |
| Name | Antonio | |
| Surname | Sapuppo | |
| Home City | Catania | |
| Phone Number | 39095229855 | Sex: Male |
| Email Address | anto@gmail.com | Country: Italy |
| Web Page | www.anto.it | Languages: English, Italian |
| Date of birth | 1998-12-02 | Interests: Sport, Looking for girls, Politics, Parties, Motorcycle |

Buttons: Back, Update

Figure A.3: Data tab

Under this tab, the whole profile of the user is shown. The user may update his profile by clicking on the *Update* command (Figure A.4).

The screenshot shows a web browser window titled 'Social Networks - Windows Internet Explorer'. The address bar shows a local file path. The main content area displays the 'Spider - Update User' form. The form is organized into two columns. The left column contains text input fields for Username (Antonio5982), Password (masked with asterisks), Re-write Password (masked), Name (Antonio), Surname (Sapuppo), Home City (Cagliari), Phone Number (+39095229884), Email Address (anto@gmail.com), Web Page (www.anto.it), and Date of birth (2/12/1998). The right column contains a radio button for Sex (Male selected), a dropdown for Country (Italy), a list of checkboxes for Languages (Danish, English, France, German, Italian, Spanish), and a list of checkboxes for Interests (Music, Sport, Cars, Looking for girls, Politics, Parties, Fashions, Motorbike). At the bottom of the form are two buttons: 'Update' and 'Back'.

Figure A.4: Update form

The update form is the same as in case of updating the profile using the mobile phone. After changing some values the user needs to click on the *Update* command to update his profile.

Friends

If the user clicks on the *Friends* tab, Figure A.5 will be shown.

The screenshot shows a web browser window titled 'Social Networks - Windows Internet Explorer'. The address bar shows a local file path. The main content area displays the 'Friends' tab. On the left, there is a tree view showing a hierarchy of users: anto5982, Paf, Beier, Kasper, wip, @we3, Tindann, oerog, Sog, Joao and maria, Morten, Juao, Joao and maria, Morten, Gabry, and Gpp. On the right, the full profile of the selected user 'Morten' is shown. The profile includes fields for Username (Morten), Name, Surname, Home City (Aalborg), Phone Number, Email Address, Web Page, Date of birth (2005-03-25), Sex (Male), Country (Denmark), Languages (Danish, English), Interests (Sport, Cars), and a 'Message' button. At the bottom, there is a 'News' section with a notification: 'morten has been accepted your invitation'. Below the notification are 'Show' and 'Delete' buttons. At the bottom of the page are 'Select' and 'Back' buttons.

Figure A.5: Friends tab - Full profile

On the left side of Figure A.5, a tree is shown. This tree contains the friends of the user logged in the web site and the friends of his friends. If the user clicks in one of his friends, the full profile of the chosen username will be shown on the top-right side of the screen. The user is able to send messages to this friend.

If the chosen username is one of the friends of his friends, but at the same time is not one of his friends, only the business card of the chosen username will be shown (Figure A.6).

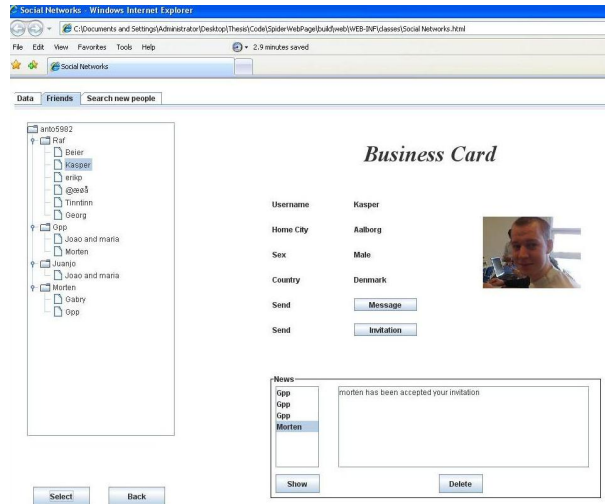


Figure A.6: Friends tab - Business card

In this example the user clicked on the username *Kasper*, which is a friend of the user name *Raf*. Kasper is not also a friend of the user logged in the web site, and therefore only his business card is shown. In this case the user is able to send either messages or invitation to him.

When the user clicks on the *Message* command in order to send a message, Figure A.7 will appear.

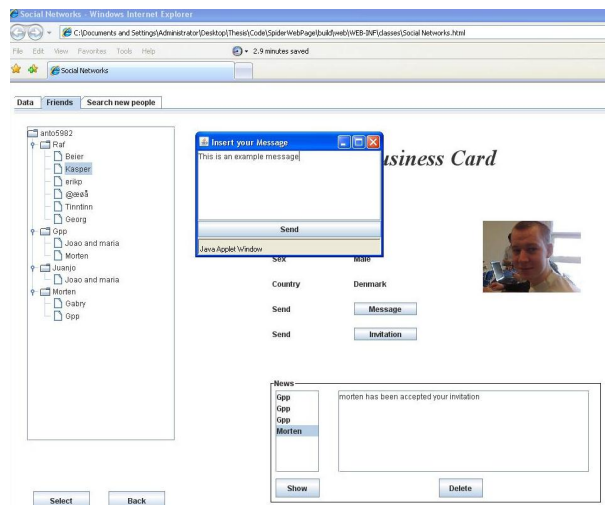


Figure A.7: Friends tab - Send a message

After typing the message, the user needs to click on the *Send* command to send the message to the chosen username.

When the user clicks on the *Invitation* command in order to send an invitation to the chosen username, Figure A.8 will appear.

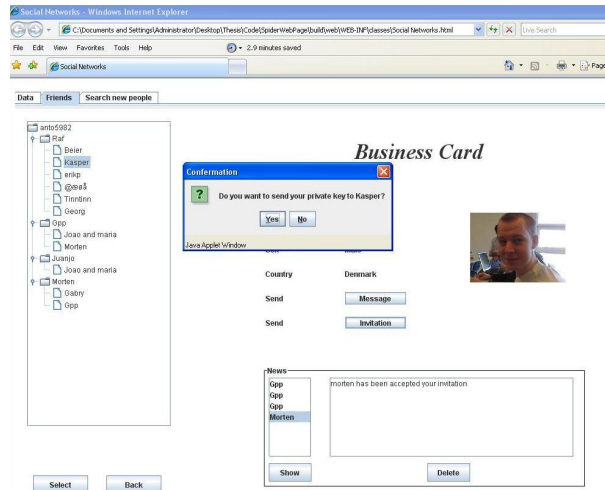


Figure A.8: Friends tab - send an invitation

To send the invitation a confirmation is needed. After clicking on *YES* button the invitation will be sent.

In the bottom-right side of the screen the news of the user containing received messages and invitations are placed. If the selected new is a message, it will be shown in the same rectangle on the screen; otherwise if the new contains an invitation, a new window will be opened asking the user whether he wants to accept or refuse the received invitation. If the user will accept it, the tree on the left side of the page will be refreshed automatically.

Search people

When the user selects this tab, Figure A.9 will appear.

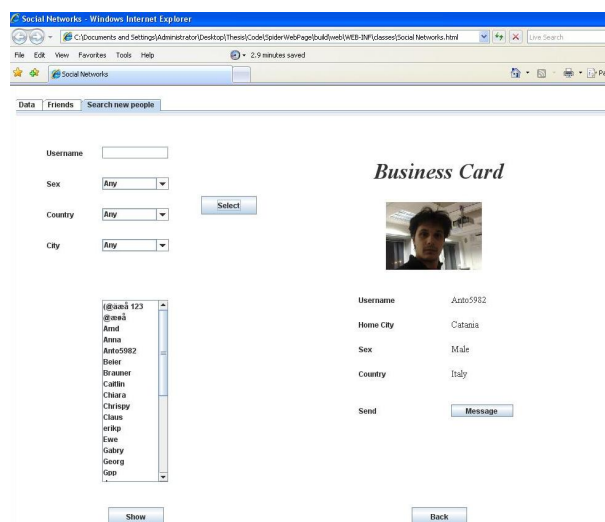


Figure A.9: Search people tab

On the top-left side of the web page, there are the filter criteria which can be applied to the search of new contacts. When the user sets the criteria and clicks on the *Select* command, the users that satisfy the applied filter criteria will be displayed in the list in the bottom-left part of the page. After selecting one of the usernames displayed in this list the user needs to click on the *Show* command. As a result, the business card of the chosen username will appear on the right side of the page. If the chosen username is a friend of the logged user, the user will

be able to send only messages to the friend. If the chosen username is an unknown person, the user will be able to send messages or invitation to the chosen username.

In Figure A.10 an example is given.

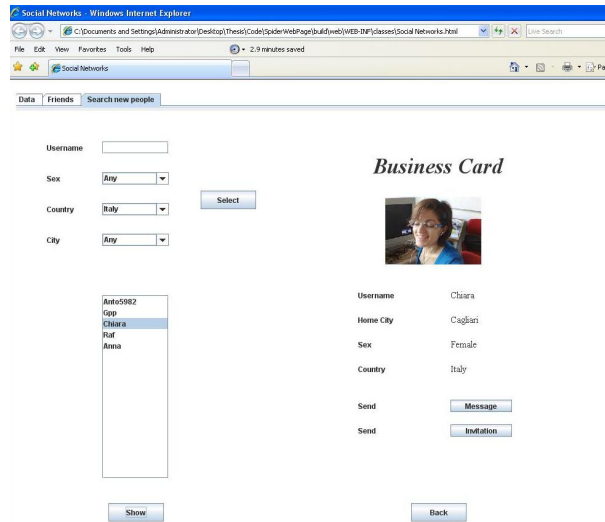


Figure A.10: Search people tab - Filter criteria applied

In this example the user selects the *Italy* filter criteria on the Country field. As a result, a list composed of five elements is shown. The user selected *Chiara* username and he clicked on the show command. The business card of *Chiara* is shown on the right side of the page. The user is able to send messages and invitation to *Chiara* due to the fact that she is not a friend of the username logged in the application.

Appendix B

Initialization and mission actions tests

In this appendix, the initialization and mission actions described in sections 5.2.1 and 5.2.2 are tested. For each action one or more test cases are defined and for each test case, the expected result, the actual result and the final result are presented.

Initialization actions

1. Register a new user action

- Test case: Inserting a username already existing in the database

Expected result

- The profile should not be inserted in the database
- The business card should not be stored in the local memory of the device

Actual result

- The profile of the user was sent to the server and it was not inserted in the database
- An error alert was displayed for three seconds, then the registration form was displayed on the screen

Result: **PASSED**

- Test case: Some mandatory parameters are not inserted

Expected result

- The profile should not be inserted in the database
- The business card should not be stored in the local memory of the device

Actual result

- An error alert was displayed on the screen for 3 seconds and then the registration form was displayed

Result: **PASSED**

- Test case: Registering a new user - all parameters are inserted correctly and the typed username was not previously inserted in the database

Expected result

- The profile of the user should be stored in the database
- The business card of the user should be stored in the local memory of the device

Actual result

- The profile of the user was sent to the server and inserted in the database
- The business card of the user was stored in the local memory of the device

- A confirmation alert was displayed on the screen for 3 seconds and then the business card of the user was displayed

Result: **PASSED**

2. Delete user action

- Test case: Typing password which doesn't match with the username

Expected result

- The username should not be deleted from the database
- The business card of the chosen username should not be deleted from the local memory of the device

Actual result

- The username and password were sent to the server and no row was deleted from the database
- An error alert was displayed on the screen for 3 seconds and then the login form was displayed

Result: **PASSED**

- Test case: Typing a valid password

Expected result

- The profile of the chosen username should be deleted from the database
- The business card of the chosen username should be deleted from the local memory of the device

Expected result

- The username and password were sent to the server
- The profile, news, friendships and invitations of the user were deleted from the database
- The business card of the user was deleted from the local memory of the device
- A confirmation alert was shown on the screen for 3 seconds and then the login form was displayed

Result: **PASSED**

3. Login as another user action

- Test case: Typing a username which does not exist in the database

Expected result

- An error alert should be shown on the screen

Actual result

- The username and password were sent to the server and no business card was received
- An error alert was displayed on the screen for three seconds and then the login as another user form was displayed

Result: **PASSED**

- Test case: Typing username and password which do not match

Expected result

- An error alert should be shown on the screen

Actual result

- The username and password were sent to the server and no business card was received

- An error alert was displayed on the screen for three seconds and then the login as another user form was displayed

Result: **PASSED**

- Test case: Typing valid username and password

Expected result

- The business card of the typed username should be stored in the local memory of the device and then shown on the screen

Actual result

- The username and password were sent to the server, the business card was received and stored into the local memory of the device
- A confirmation alert was displayed for three seconds and then the business card of the typed username was shown on the screen

Result: **PASSED**

4. Login action

- Test case: Typing wrong password

Expected result

- An error alert should be shown

Expected result

- An error alert was displayed on the screen for three seconds and then the login form was shown

Result: **PASSED**

- Test case: Inserting correct password

Expected result

- The business card of the chosen username should be displayed on the screen

Expected result

- The business card of the chosen username was shown on the screen

Result: **PASSED**

Mission actions

1. My profile action

- Test case: Showing the profile of the user

Expected result

- The profile should be displayed on the screen

Actual result

- The username of the user logged in to the application was sent to the server, the profile of the user was received and displayed on the screen

Result: **PASSED**

- Test case: Updating the profile of the user

Expected result

- The changed parameters should be updated in the database and in the local memory of the device

Actual result

- The updated profile of the user was sent to the server and it was updated in the database

- A confirmation alert was displayed for three seconds and then the updated business card was shown on the screen

Result: **PASSED**

- Test case: Showing the full screen picture of the user

Expected result

- The full screen picture of the user should be displayed on the screen

Actual result

- The full screen picture of the user was displayed on the screen after performing the My picture command

Result: **PASSED**

2. My friends

- Test case: Showing friends of the user

Expected result

- All the friendships of the user stored in the database should be displayed in the list of friends

Expected result

- The username was sent to the server and the list of friends was received
- All the friends present in the database were displayed in the list of friends

Result: **PASSED**

- Test case: Showing the profile of a friend

Expected result

- The full profile of the chosen username should be displayed on the screen

Expected result

- The chosen username was sent to the server, his full profile was received from the server and displayed on the screen

Result: **PASSED**

3. Retrieve news action

- Test case: Showing the news of the user

Expected result

- The number of the news of the user stored in the database should be the same as the number of elements in the list of news

Actual result

- The username was sent to the server and the usernames of the senders of all news were received and displayed on the screen
- The number of the news displayed on the screen was the same as the number of the news registered in the database

Result: **PASSED**

- Test case: Reading a message

Expected result

- The message should be displayed on the screen

Actual result

- The news of the user were displayed on the screen, one of them was selected and its identification number was sent to the server

- The text message corresponding to its identification number in the database was received and displayed on the screen

Result: **PASSED**

- Test case: Accepting an invitation

Expected result

- The friendship between the sender of the invitation and the user should be stored in the database and in the local memory of the device
- After accepting the invitation the new has to be deleted automatically from the database and from the list of news

Actual result

- One of the news including the invitation message was selected
- An information alert asking whether the user wants to accept or refuse the invitation was displayed and accept option was selected
- The accepted invitation message was sent to the server and the friendship between the sender of the invitation and the user was inserted in the database
- The new was deleted from the database and a message stating that the invitation was accepted was inserted in the database
- The friendship between the sender of the invitation and the user was stored in the local memory of the device and the new was deleted from the list of news
- A confirmation alert was displayed for 3 seconds and then the list of news was shown on the screen

Result: **PASSED**

- Test case: Refusing an invitation

Expected result

- After refusing the invitation the new should be deleted automatically from the database and from the list of news

Actual result

- One of the news including the invitation message was selected
- An information alert asking whether the user wants to accept or refuse the invitation was displayed and the refuse option was selected
- The refused invitation message was sent to the server and the new was deleted from the database
- A message stating that the invitation was refused was inserted in the database
- The new was deleted from the list of news, then a confirmation alert was displayed for 3 seconds and afterwards the list of news was shown on the screen

Result: **PASSED**

- Test case: Deleting a message

Expected result

- The new should be deleted from the database and from the list of news

Actual result

- One of the news including a message was displayed on the screen
- The delete command was performed and the identification number of the new was sent to the server
- The new was deleted from the database and from the list of news
- A confirmation alert was displayed for 3 seconds and then the list of news was shown on the screen

Result: **PASSED**

- Test case: Reply message

Expected result

- The message should be inserted in the database

Actual result

- One of the news including a message was displayed on the screen
- The Reply command was selected and a message was typed
- The reply message was sent to the server and inserted in the database
- A confirmation alert was displayed for 3 seconds and then the list of news was shown on the screen

Result: **PASSED**

4. Search new contacts

- Test case: Applying filter criteria

Expected result

- After applying filters criteria, the list of usernames sent back from the server should satisfy the requirements

Actual result

- After typing the pseudo username and/or selecting the gender and/or the city and/or the country, these selections were sent to the server
- The server returned a list of usernames satisfying the filter criteria

Result: **PASSED**

- Test case: Sending message or invitation

Expected result

- The message or the invitation should be stored in the database

Actual result

- The text message or the invitation was sent to the server with the username of the sender and the username of the receiver
- The message or the invitation was inserted in the database
- A confirmation alert was displayed for 3 seconds and then the business card of the user was shown on the screen

Result: **PASSED**

5. Synch with the server

- Test case: Performing the Synch with the server action

Expected result

- After performing this action, the business card of the user and the list of his friends should be the same as the ones stored in the database

Actual result

- The username and the list of the friends stored in the local memory were sent to the server
- The business card and the list of the friends stored in the database were received from the server
- A confirmation alert was displayed for 3 seconds and the business card of the user was shown
- The business cards and the lists of friends stored in the database and in the local memory of the device were the same after this operation

Result: **PASSED**

Appendix C

My virtual world action tests

In this section the main tests regarding the My virtual world actions are presented. For each test case, the expected result, the actual result and the final result are presented.

Help action

- Test case: Showing the help menu
- Expected result
 - When the user approaches the barmaid the help menu should be displayed
- Actual result
 - The help menu was displayed after approaching the barmaid
- Result: **PASSED**

Chat action

- Test case: Displaying the chat form
- Expected result
 - When the user approaches the girl seated at the table and selects the Chat command, the chat form should be displayed
- Actual result
 - The chat form was displayed after approaching the girl seated at the table and selecting the Chat command
- Result: **PASSED**

Seach people action

- Test case: Performing the search people action
- Expected result
 - City and country fields should be fullfilled with distinct values of the users present within the Bluetooth range
 - After applying some filter criteria the obtained list of users should satisfy the requirements
- Actual result

- The city and countries fields were fulfilled with distinct values of the virtual people present within the virtual world of the user
- After applying some filter criteria the obtained list of users was satisfying the previous selections
- Result: **PASSED**

Waiting list

- Test case: Displaying and applying the waiting list
- Expected result
 - The number of friends should be the same as the number of elements of the waiting list
 - If an element of the waiting list is selected, this element should be represented with the searched figurine and the waiting list icon should be activated
- Actual result
 - The number of the friends stored in the local memory of the device was the same as the number of elements of the waiting list
 - After selecting an element of the waiting list, this element was represented with the searched figurine and the waiting list icon was activated.
- Result: **PASSED**

Connect to the right service

- Test case: Sending a message
- Expected result
 - Among six Spider applications, the message should be sent to the selected receiver
- Actual result
 - The message was sent to the selected receiver
 - A confirmation message was displayed and then the user interface was shown on the screen
- Result: **PASSED**

Send multiple messages

- Test case: Sending five messages to the same receiver in the row
- Expected result
 - All the messages should be sent correctly
- Actual result
 - For each message a confirmation alert was displayed
 - All the messages were received correctly
- Result: **PASSED**

Receive multiple message

- Test case: Eight Spider applications send a message to the same receiver in a very short period of time
- Expected result
 - The receiver should get all the messages
- Actual result
 - All the messages were received
- Result: **PASSED**

Send an accepted invitation message

- Test case: The receiver is in the range of the user
 - Expected result
 - After sending an accepted invitation message the figurines of the sender and the receiver should change from the status of unknown people to the status of friends. Additionally, the friends icon should be activated
 - Actual result
 - The accepted invitation message was sent and received correctly. The figurines of the receiver and the sender changed from the status of unknown people to the status of friends. Additionally, the friends icon was activated in both devices
 - Result: **PASSED**
- Test case: The receiver is not in the range of the user
 - Expected result
 - After sending an accepted invitation message the application should notify the sender that the receiver is not anymore in his Bluetooth range
 - The message should be stored in the local memory of the device
 - Actual result
 - The application displayed an error alert stating that the user was not anymore in its Bluetooth range
 - The message was stored in the local memory of the device
 - When the receiver of the accepted invitation message appeared in the virtual world of the user, the message was sent and the figurines of the receiver and the sender changed from the status of unknown people to the status of friends. Additionally, the friends icon was activated in both devices
 - Result: **PASSED**

Refresh

- Test case: Among 8 devices only 5 have the Spider application running on
- Expected result
 - The number of found devices should be 5
- Actual result
 - The number of found people was 5
- Result: **PASSED**

Exit

- Test case: Performing the exit action
- Expected result
 - The device should not be discoverable
- Actual result
 - After performing the exit action the device was not discoverable anymore
- Result: **PASSED**

Appendix D

Web page of the project and Questionnaire

Web page of the project

The following url is the web page of the project:

http://mobiledevices.kom.aau.dk/projects/student_projects/spring_2007/social_network/

The web page contains:

- Spider.jar: to install the application on the mobile phone
- Spider user guide: a full guide of Spider
- Spider F.A.Q.: to help the students to perform all the actions
- Known issue of Spider

A screenshot of this web page is presented in Figure D.1.

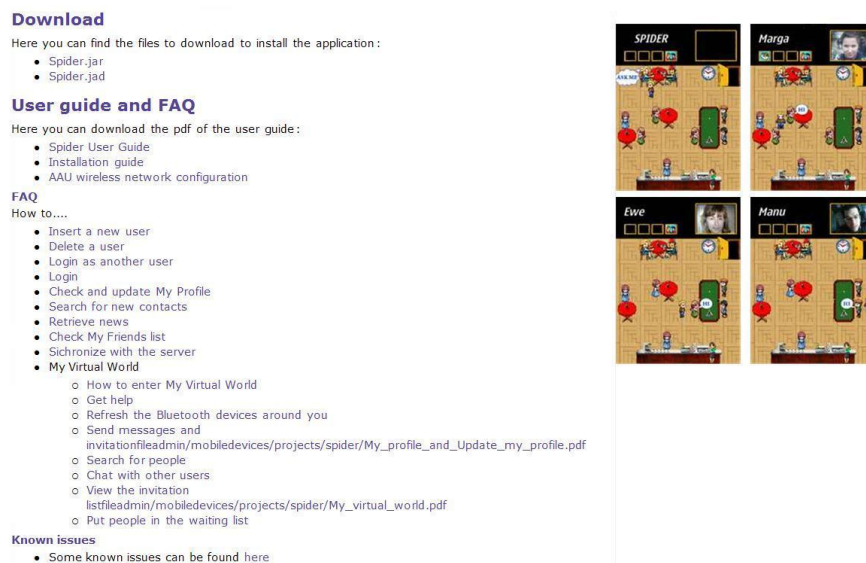


Figure D.1: Project web page

Questionnaire: usability test

- Did you manage to perform this action?

Yes No

If you didn't manage to perform this action, please use the guide.
If you used the guide to perform this action, please answer the following question.

- How helpful was the guide?

Very helpful Helpful Not clear

If you have performed this action successfully (with or without using the guide), please answer the following questions:

- How difficult it was to perform this action?

Very easy Easy Medium difficult Hard Very hard

- Did you encounter any problems to perform this action?

Yes No

– **If YES**, was this problem mentioned in the known issues?

Yes No

* **If it was not in the known issues** please report it here

- Do you have any comments regarding this action?

Questionnaire: General evaluation

- How difficult it was to use Spider?
 Very easy Easy Medium difficult Hard Very hard
- How good is the the interaction with the user?
 Very easy Easy Medium difficult Hard Very hard
- How good is the game flow of the application?
 Very easy Easy Medium difficult Hard Very hard
- How do you evaluate the idea of the application?
 Excellent Very good Good Sufficient Bad
- How do you evaluate the My virtual world idea?
 Excellent Very good Good Sufficient Bad
- How do you evaluate the HTTP actions in terms of fastness and the robustness?
 Excellent Very good Good Sufficient Bad
- Do you think that this application is an innovative idea? Motivate your answer
- Write 3 things that you liked
- Write 3 things that you consider unnecessary
- Write 3 things that can be improved

Appendix E

UML

Internet connectivity

The UML presented in Figure E.1 depicts the idea of the application design regarding the Internet connectivity part. The responsibilities of each class are described below.

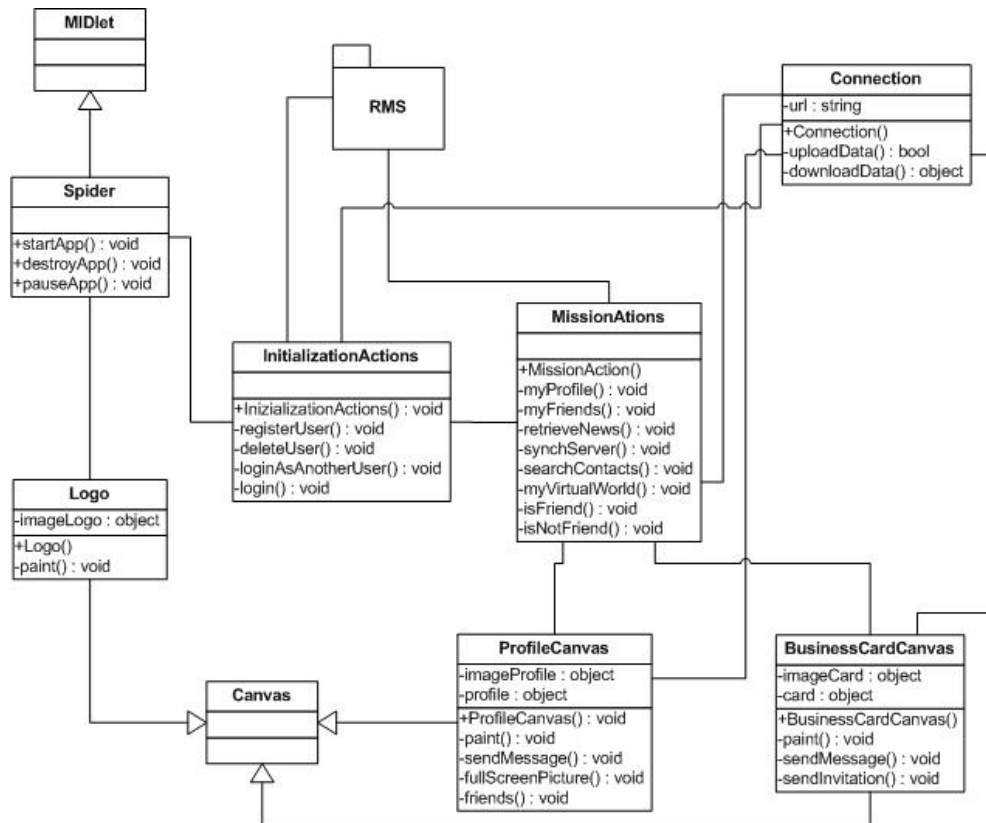


Figure E.1: Class diagram - Internet connectivity

The Spider class is the main class of the application. It extends the MIDlet class which provides the startApp method to launch the application.

The Spider class instantiates the InitializationActions class, which provides the following main methods:

- registerUser: to register a user in the application
- deleteUser: to delete a user from the application

- loginAsAnotherUser: to download the business card of the user into the local memory of the device
- login: to login into the application

A detailed description of these actions is given in section 5.2.1.

When the user is logged in the application, the MissionAction class is instantiated. This class provides the following main methods:

- myProfile: to view the profile of the user
- myFriends: to view the list of friends of the user
- retrieveNews: to view messages and invitations received by the user
- synchServer: to synchronize with the server
- searchContacts: to search new contacts already registered in the database
- myVirtualWorld: to start the My virtual world action

A detailed description of these actions is given in section 5.2.2. This class is linked with the ProfileCanvas and BusinessCardCanvas classes which extend the Canvas class.

The BusinessCardCanvas is in charge of displaying the business card of the user on the screen. The ProfileCanvas is responsible of showing the full profile of the user. These classes are also used to view business cards or profiles of other users. The application instantiates the BusinessCardCanvas class if the chosen username is not a friend of the user, otherwise it instantiates the ProfileCanvas class.

The BusinessCardCanvas class provides the following methods:

- the paint method to draw the business card of the chosen username on the screen
- the send message method, in order to send a message to the chosen username
- the fullScreenPicture method, in order to see the picture of the chosen username resized to fit the full screen of the mobile device

The ProfileCanvas class provides the following methods:

- the paint method to draw the full profile of the chosen username on the screen
- the send message method, in order to send a message to the chosen username
- the sendInvitation method, in order to send an invitation to be friends to the chosen username

The Connection class is in charge of sending the data to the server using the internet connectivity. This class is linked with the MissionActions, InitializationActions, ProfileCanvas and BusinessCardCanvas classes. The connection class is composed of two main methods:

- uploadData: to send the data which need to be inserted in the database
- downloadData: to download data from the server

Additionally, MissionActions and InitializationActions classes make use of the RMS package to store and retrieve data from the local memory of the device. A description of the way in which the classes manage to store and retrieve data from the local memory of the device is given in section 5.1.

Bluetooth connectivity

The UML presented in Figure E.2 depicts the idea of the application design regarding the Bluetooth part of the project. The responsibilities of each class are described below.

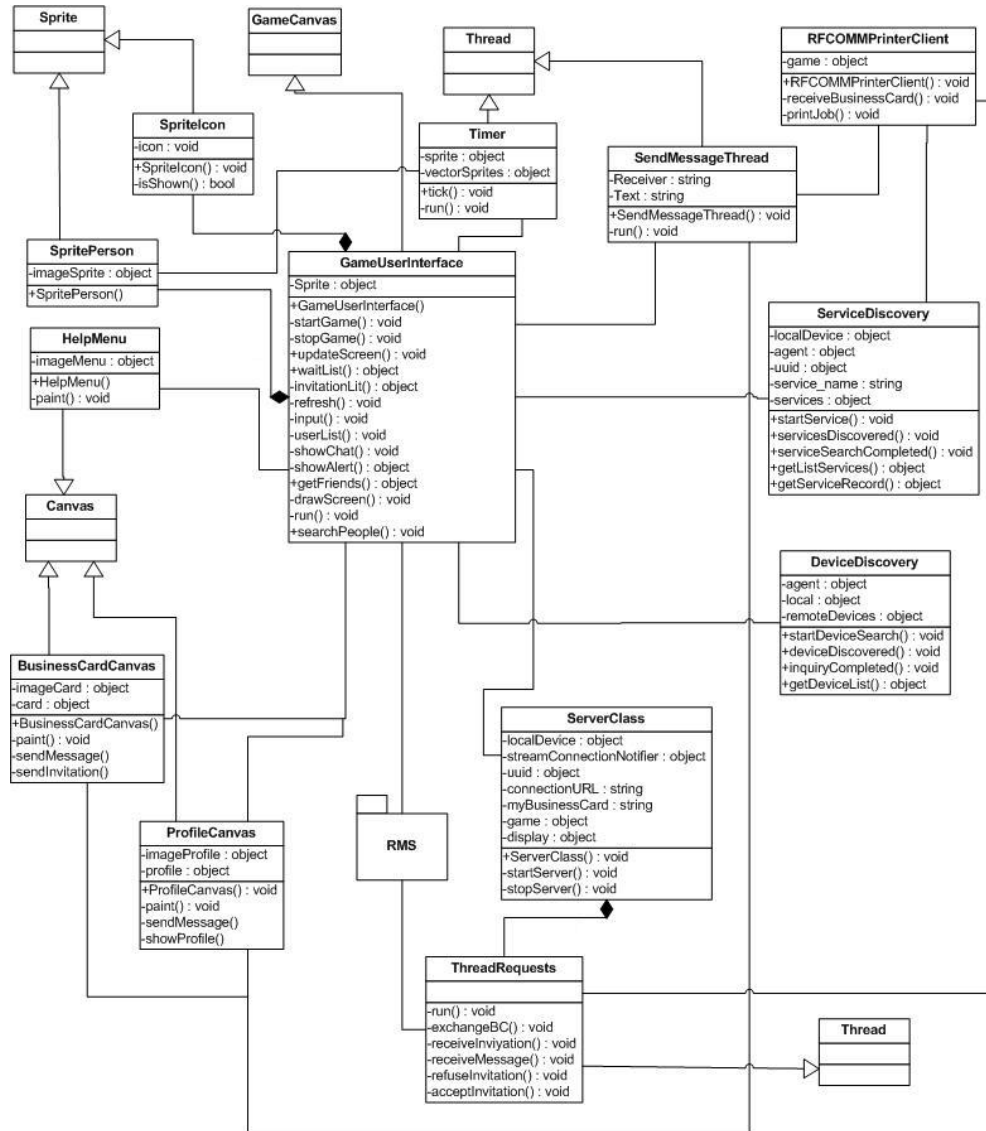


Figure E.2: Class diagram - Bluetooth connectivity

When the user performs the My virtual world action the ServerClass and GameUserInterface classes are instantiated.

The ServerClass class is in charge of rendering the Spider application discoverable for other Spider applications and of accepting requests from clients. A detailed description of the ServerClass class is given in section 6.2.1. The serverClass contains the startServer and stopServer methods in order to make the device respectively discoverable or undiscoverable for other Spider applications.

Each time the ServerClass receives a request from a client, it instantiates the ThreadRequests class. This class takes care of the request of the client and makes use of the RMS package to store and retrieve data from the local memory of the device. The following methods are implemented in the ThreadRequests class:

- exchangeBC: in order to exchange the business cards between the discoverer device and

the discovered device

- receiveInvitation: this method is invoked when the user receives an invitation from another user
- receiveMessage: this method is invoked when a message from another user is received
- refuseInvitation: this method is invoked when another user refuses an invitation sent by another user previously
- acceptInvitation: this method is invoked when another user accepts an invitation sent by the user previously

These actions are described in details in the section 6.3.2.

The GameUserInterface class extends the GameCanvas class. This is the main class of the Bluetooth connectivity. This class has the responsibilities of control the user interface of the application. When this class is instantiated, a Sprite object is also instantiated. This object represents the virtual person inside the screen of the mobile phone. Moreover, this class makes use of the RMS package in order to store and retrieve data from the local memory of the device. GameUserInterface is composed of the following methods:

- startGame: this method is invoked to start the game of the application
- stopGame: this method is invoked to stop the game of the application
- drawScreen: this method is in charge of painting on the screen the user interface of the application
- input: this method is in charge of capturing the sporadic events of the user. It means that if the user would like to move the virtual person to the right, this event is captured by this method
- waitList: this method is used to perform the waiting list action
- invitationList: this method is used to perform the invitation list action
- userList: this method is invoked to show on the screen the list of users that are present within the Bluetooth range of the user
- showChat: this method is invoked to show messages sent and received through the Bluetooth technology
- searchPeople: this method is used to perform the search people action
- updateScreen: this method is invoked when a device is discovered and a new virtual person should be shown on the screen of the application
- refresh: this method is invoked when the user would like to start a Bluetooth discovery

The main methods of this class are described in details in section 6.3.1.

The Timer class is in charge of moving virtual people in the screen. This timer is used to move the virtual person representing the user and other virtual people within the virtual world from their initial position in the screen to their final positions. A detailed description of the timer is given in section 6.1.2.

The SpriteIcon and SpritePerson classes represent respectively the icons in the black zone of the screen and the virtual people present in the active zone. A detailed description of the design of the screen is given in section 6.1.1.

The HelpMenu class is in charge of showing the help menu when the user performs the help action. A detailed description of the help action is given in section 6.3.1.

The BusinessCardCanvas and ProfileCanvas classes are the same classes as the ones in the scheme presented in the previous section.

The DeviceDiscovery and ServiceDiscovery classes are responsible for refreshing the screen of the mobile phone. They start a Bluetooth inquiry and return the devices within the Bluetooth range of the user equipped with the Spider application. Each time a new Spider application is found, the receiveBusinessCard method of the RFCOMMPrinterClass is invoked to exchange the business cards between the discovered and discoverer devices.

The DeviceDiscovery class is composed of the following methods:

- startDeviceSearch: to start the device discovery
- deviceDiscovered: this method is invoked each time the device discovery finds a Bluetooth device
- inquiryCompleted: it is invoked when device discovery ends
- getDeviceList: it is invoked when the list of found devices is required

The ServiceDiscovery class is composed of the following methods:

- startService: it is invoked when a device has to be analyzed to check if it is equipped with the desired service
- servicesDiscovered: it is invoked when an inspected device possesses the desired service
- serviceSearchCompleted: it is invoked when the service discovery ends
- getServiceRecord: it is invoked when the url address of a specific device is required

A detailed description how the device and service discoveries work is given in section 6.2.

In order to send data the application uses the SendMessageThread class. This class extends the Thread class and takes as input parameters the receiver and the sender of the message and the text of the message. In order to send the message the printJob method of the RFCOMM-PrinterClient is invoked.

Bibliography

- [1] Forum Nokia. *Bluetooth Technology Overview*. 2003.
- [2] Sun Microsystems. *Sun Java System Application Server Platform Edition 9.0*. 2006.
- [3] *MySQL vs PostgreSQL*. <http://articles.techrepublic.com>.
- [4] *An Overview of MySQL*. <http://media.wiley.com>, 2003.
- [5] Symbian. *Ownership*. <http://www.symbian.com/about/overview/ownership/ownership.html>.
- [6] Sun Microsystems. *Java EE 5 SDK Preview and Sun Java System Application Server Platform Edition 9 Beta: A Feature Summary*. <http://developers.sun.com/appserver>.
- [7] *Application Server*. <http://www.service-architecture.com>.
- [8] Sun Microsystems. *About Application Server Security*. <http://docs.sun.com/app/docs/doc/819-3658/6n5s5nkmo?a=view>.
- [9] *Introduction to SSL*. <http://www.tcxa.com.cn/technology/security/ssl.html>.
- [10] Russell J.T.Dyer. *MySql in a nutshell*. 2005.
- [11] MySQL. <http://dev.mysql.com/doc/refman/5.0/en/what-is-mysql.html>.
- [12] Vikram. *MySQL: The Complete Reference*. 2004.
- [13] Ryan Bannon Alvin Chin Faryaz Kassam Andrew Roszko. *MySQL Conceptual Architecture*. 2002.
- [14] Apple. *Bluetooth Architecture*. <http://developer.apple.com/documentation>.
- [15] Martin de Jode. *Programming Java 2 Micro Edition on Symbian OS*. 2004.
- [16] Fitzek Reichert. *Mobile Phone Programming and its Application to Wireless Networking*. 2007.
- [17] Sun Microsystems. *The K virtual machine (KVM)*. <http://java.sun.com/products/cldc/wp>.
- [18] Riggs Taivalsaari Peursem Huopaniemi Patel Uotila. *Programming Programming Wireless Devices with the Java 2 Platform Micro Edition, Second Edition*. 2003.
- [19] Sun Microsystems. *Connected Limited Device Configuration (CLDC)*. <http://java.sun.com/products/cldc>.
- [20] Sun Microsystems. *Mobile Information Device Profile*. <http://java.sun.com/products/midp/>.
- [21] Sun Microsystems. *Java ME Technology APIs & Docs*. <http://java.sun.com/javame/reference/apis.jsp>.
- [22] Graham Hamilton Rick Cattell Maydene Fisher. *JDBC Database access with Java*. 1997.
- [23] Sun Microsystems. *Servlet*. http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets2.html.

- [24] Nokia. *Enabling Bluetooth Limited Inquiry (LIAC)*. <http://wiki.forum.nokia.com>.
- [25] Sourceforge. *A J2ME Unit Testing Framework*. <http://j2meunit.sourceforge.net>.
- [26] Nokia. *S60 Platform: Bluetooth API Developer's Guide*. 2006.
- [27] Steve Babin. *Developing Software for Symbian OS*. 2006.
- [28] Apple. *BTBluetoothBasics*.
- [29] Riku Mettälä. *Bluetooth Protocol Architecture v1.0*. 1999.
- [30] AU-System. *Bluetooth White Paper 1.1*. 2000.
- [31] Sun Microsystems. *Sun Java System Application Server Platform Edition 9 Administration Guide*. 2006.
- [32] Lars Kirkhus Anders R. Sveen. *An examination of mobile devices for spontaneous collaboration*. 2003.
- [33] Perrucci Gian Paolo Ferri Maurizio. *SMARTEX*. 2005.
- [34] Sun Microsystems. *Java ME at a Glance*. <http://java.sun.com/javame>.