VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Saeed Ahmad

Developing a Bluetooth Ordering Application

Technology and Communication
2011

ACKNOWLEDGEMENTS

First I would like to thank to God almighty who helped me and my parents who prayed for me, then I like to thank to my supervisor for helping me to improve the application. I like to thank also to my wife who supported me and always kept a calm attitude in situations when I was stuck in development due to some error. The next day was easy to find solution with fresh mind. I like to thank my friends too who gave me encouragement.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Technology and Communication

ABSTRACT

Author	Saeed Ahmad
Title	Developing a Bluetooth Ordering Application
Year	2011
Language	English
Pages	47 + 1 Appendices
Name of Supervisor	Ghodrat Moghadampour

The Java Platform, Micro Edition (J2ME) is a widely adopted technology to, in a controlled way, introduce third party developers to the otherwise inaccessible platforms in mobile phones, and at the same time allows applications to run on devices from different manufacturers and Java APIs for Bluetooth Wireless Technology (JABWT) provide the basic structure for developing Java Bluetooth applications . It also provides how different Bluetooth actions like inquiry and service discovery are done with the Java API. Code samples are also included highlighting the functionality available in JABWT. Connection to MySQL database using Netbeans 6.9 is described as well. Pizza Restaurant Application is wirelessly deployed on the users mobile whenever he enters the specified area, such as the restaurant, enabling him to perform activities such as to see the menu and rate list of pizzas, and to place order or to place complains or suggestions for managers. The administrator of the application is entitled to see the list of orders, to change the prices of food and to maintain the services provided by the management. The customer can only make payment in cash. There was a limitation at client's side when getting database records from pizza restaurant server to display rate list and menu, because only few libraries are available in J2ME for small devices. But discussing with the supervisor separate functions for reading and displaying multiple records from the server were developed.

Keywords Bluetooth implementation in J2ME

CONTENTS

LISTS OF PICTURES

1 INTRODUCTION

Electronic commerce, known as e-commerce or ecommerce, includes the buying and selling of products or services over electronic systems such as the Internet and other computer networks. The amount of trade carried out electronically has grown extraordinarily since the spread of the Internet.

A wide variety of commerce is carried out in this way, drawing on innovations in electronic funds, supply chain management, Internet marketing, online transaction processing, interchange of electronic data, inventory management systems, and automated data collection systems. Modern electronic commerce uses the World Wide Web at some point in the transaction's lifecycle, although it can cover even a wider range of technologies such as e-mail.

A sharp increase in the use of mobile applications and mobile devices within enterprises means organizations will need a better way to manage their mobility going forward, a recent study found.

Mobile software is designed to run on handled computers, personal digital assistants (PDAs), enterprise digital assistant (EDAs), smart phones and cell phones.

Recent model cell phones have included the ability to run user-installed software. Mobile development lists the differences among the various mobile softwares.

Areas of Mobile Application

- Mobile optimized search engines
- Mobile question and answer services
- Mobile directory search
- Mobile discovery services
- Mobile navigation services

1.1 Objectives

The main objective of this topic was in the beginning to implement J2ME via Bluetooth technology , to develop a software related to pizza shop and to understand these technologies. And to make client server data transfer capabilities in software. The Pizza Restaurant Application is wirelessly deployed on the users mobile whenever he enters the specified area, such as the restaurant, enabling him to perform activities such as to see the menu, and to place order. The administrator of the application is entitled to see the list of orders, to change the prices of food and to maintain the services provided by the management. The customer can only make payment in cash.

2 RELEVANT TECHNOLOGIES

In this part I have described java technologies used to develop this application. This application runs on desktop system at server side and on mobile devices at client side that involves use of J2SE and J2ME vice versa and java Bluetooth API for Bluetooth functionality on both sides.

2.1 J2SE

Java Platform Standard Edition is popularly used in Java language programming. This is the platform commonly used to create portable applications that run on server and desktop systems spanning most operating systems. J2SE features:

- Portability
- JDBC API for database access
- CORBA technology
- Java security

J2SE consists of two components: Core Java and Desktop Java. Core Java provides back-end functionality, while Desktop Java provides GUI (Graphical User Interface) functionality. J2SE contains both the J2SE Development Kit (JDK) and the Java Runtime Environment (JRE).

2.2 J2ME

Java 2 Micro Edition combines a resource constrained JVM and a set of Java APIs for developing applications for mobile devices. Device manufacturers install and pre-package there device with this JVM and associated APIs. J2ME can be divided into three parts /3/

- Configuration
- Profile
- Optional Packages

Configuration contains the JVM and some class libraries; profile is build on top of these class libraries that provides a useful set of APIs. Optional Package is an optional set of APIs which may or may not be used in development of application.

The most popular configurations and profiles of J2ME are Mobile Information Device Profile (MIDP) and Connected Limited Device Configuration (CLDC).

2.2.1 Connected Limited Device Configuration (CLDC)

The Connected Limited Device Configuration (CLDC) has been designed for resource-constrained devices like mobile phones, pagers, and mainstream personal digital assistants. It defines the base set of application programming interfaces and a virtual machine. CLDC can be used for writing applications for small devices. But it gives very limited functionality. There is no way to provide a graphical user interface, unless we use a profile to build an effective application. The only profile so far available for small devices is the Mobile Information Device Profile (MIDP). Both provide a solid Java platform for developing applications to run on devices with limited memory, processing power, and graphical capabilities, since the MID profile complements the CLDC configuration by minimising both power and memory required for limited devices. /3/

2.2.2 Mobile Information Device Profile (MIDP)

Mobile Information Device Profile (MIDP) is a specification published for the use of Java on embedded devices such as mobile phones and PDAs. MIDP is part of the Java Platform, Micro Edition (Java ME) framework and sits on top of Connected Limited Device Configuration, a set of lower level programming interfaces.

The Mobile Information Device Profile (MIDP) allows writing downloadable applications and services for network-connectable mobile devices. When combined with the Connected Limited Device Configuration (CLDC), MIDP is the Java runtime environment for compact mobile information devices, such as cell phones

and mainstream PDAs. /3/

2.2.2.1 MIDP Technology

Each Java technology has a reference implementation (RI), an API specification and a technology compatibility kit (TCK) associated to it.

- **Specification**

  MIDP 2.0

  MIDP 2.0 is a revised version of the MIDP 1.0 specification, and includes new features such as greater connectivity, an enhanced user interface, over-the-air (OTA) provisioning, multimedia and game functionality, 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.

- **Reference Implementation**

  MIDP RI 2.0

  The MIDP RI is targeted at device manufacturers who want to port this J2ME profile to another platform.

  - **Technology Compatibility Kit**

    The CLDC TCK can be licensed from Sun to verify a CLDC implementation on an exclusive platform.

  - **Development Tools**

    Sun Java Wireless Toolkit offers complete development support for developing MIDP applications in combination with today's leading assimilated development environments (IDEs.)

NetBeans Mobility Pack is a Java Integrated Development Environment (IDE) for developing applications that can be installed to Java technology-enabled mobile devices.

2.3 Java APIs for Bluetooth Wireless Technology

While Bluetooth hardware has progressed, there has been no standardized way to develop Bluetooth applications - until JSR 82 came into play. It is the first non-proprietary standard for developing Bluetooth applications by using the Java programming language. It hides the complexity of the Bluetooth protocol stack behind a set of Java APIs that allows to focus on application development rather than the low-level details of Bluetooth. JSR 82 is based on version 1.1 of the Bluetooth Specification. /1/

Like all JSRs, the Java APIs for Bluetooth are also developed through the Java Community Process. Its expert group has members representing 20 companies.

JSR 82 has two optional packages: the core Bluetooth API and the Object Exchange (OBEX) API. The latter is transport-independent and can be used without the former.

3 REQUIREMENTS ANALYSIS

In this part I have described application in terms of hardware, performance, security, quality, safety, functional requirements and implementation constraints.

3.1 Bluetooth System Requirements

The Java APIs for Bluetooth target devices with the following characteristics:

- (application memory requirements are additional)
- Bluetooth wireless network connection
- Compliant 512K minimum of total memory available (ROM and RAM) implementation of the J2ME Connected Limited Device Configuration (CLDC)

The underlying Bluetooth system upon which the Java APIs will be built must also meet certain requirements:

- The underlying system must be "competent," in compliance with the Bluetooth Qualification Program, for at least the Generic Access Profile, Service Discovery Application Profile, and Serial Port Profile.
- The system must hold three communication layers as defined in the 1.1 Bluetooth Specification, and the implementation of this API must have access to them: (SDP), Radio Frequency Communications Protocol (RFCOMM), Logical Link Control and Adaptation Protocol (L2CAP) and Service Discovery Protocol.
- The system should provide a Bluetooth Control Center (BCC), a control panel almost similar to the application that allows a user to define specific values for certain configuration parameters in a stack.

OBEX support should be provided in the fundamental Bluetooth system or by the implementation of the API. The OBEX protocol provides assistance for object

exchanges, and forms the basis for Bluetooth profiles such as the Synchronization Profile and the File Transfer Profile. /1/

3.2 Performance Requirements

The performance requirements of our systems are:

- Our system should provide 24 hours service for ease of customers.
- High response times are expected of the system.
- If the system has been stuck due to some reasons the system should be able to regain its original state in minimum time.
- Pizza Shop Application is designed such that the system settings are not disturbed by its usage.

3.3 Safety Requirements

The safety requirements for our system are:

- Special protection must be provided against harmful external agents such as viruses, key loggers, and hackers.
- If some data infected with viruses succeeds to enter the system, then effected files should not be able to upload.

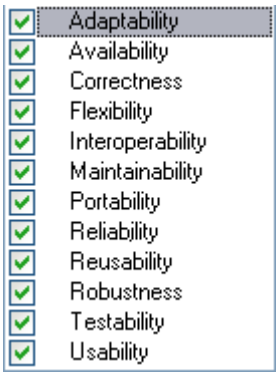
3.4 Security Requirements

The security requirements for our system are:

- Users are allowed to access the system by Mobile application only. Extra privileges are given to personnel such as managers and owners.
- All the data that is being entered should be scanned through an anti-virus scheme.
- No person other than the manager should have the right to access database and change the information or data residing there.

3.5 Software Quality Measures

The following are some software quality attributes of our system:



3.6 Functional Requirements

- The manager will be able to cancel or clear an order or update the menu list. Functional requirement to enforce this rule is login through a user name and a password in the operating system.

- The user will only be able to see the information relating the restaurant that is permitted by the authorities, like menu list, the price list, etc. Also, the user is permitted to change his order within 5 minutes of the prior placement.

3.7 Design and Implementation Constraints

Design and implementation constraints are:

- The developers should keep in mind the memory constraint for RAM (not less than 512MB) and hard disks (40GB recommended) for the server system.
- For our ease we shall be using Unified Modeling Language, because we are well familiar of this. This is a constraint on our designing procedure not to use language other than UML.
- The development platform is Java.
- The IDE used for the development of the application must be NetBeans IDE 6.9.

- As we have the database in MYSQL, so the application needs to be developed accordingly. /2/
- The credit card verification is not implemented in this version.

3.8 Assumptions and Dependencies

- Our system depends on rules and regulations like database licensing.
- The system is supposed to provide 24-hr service.
- The users are assumed to have mobile phones with the Bluetooth capability enabled.

4 APPLICATION DESCRIPTION

This figure describes how this application functions in actual working environment. First level shows that Bluetooth uses radio signals to communicate with other Bluetooth device. In 2nd level the operating system checks and installs Bluetooth hardware driver and after that our application detects Bluetooth stack in underlying system. In the next level the Connected Limited Device Configuration (CLDC) makes the base set of application programming interfaces and K virtual machine is a java virtual machine which provides the basis for the CLDC reference implementation. MIDP is the Java runtime environment for cell phones in our application. Bluetooth API (Bluecove 2.1) conceals the complexity of the Bluetooth protocol stack behind a set of Java APIs that allows us to concentrate on application development rather than the low-level details of Bluetooth. At top level we are interacting with the application using graphical user interfaces at the client and the server side in pizza restaurant application.

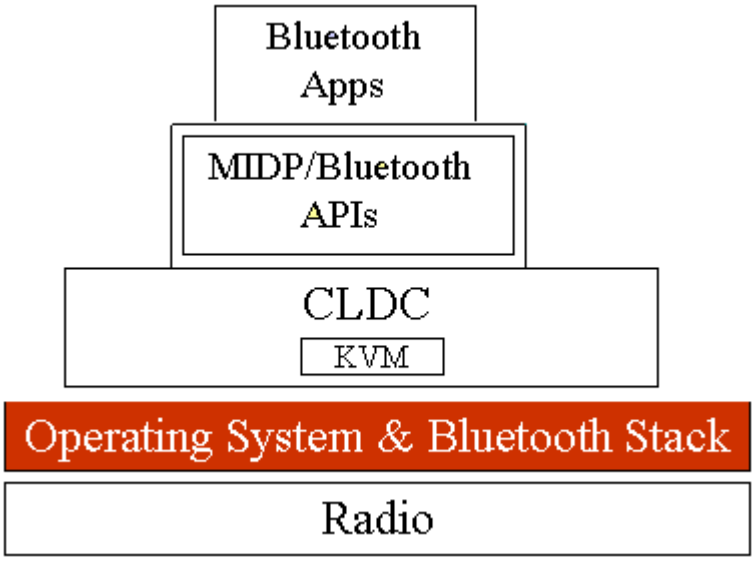


Figure 3. Basic Architecture of Bluetooth followed in our Application./1/

4.1 Bluetooth Control Center (BCC)

Bluetooth devices that execute this API may allow multiple applications to perform concurrently. The BCC is a set of capabilities that allow a user or OEM to solve contradictory application requests by defining precise values for certain configuration parameters in a Bluetooth stack. BCC is the central authority for local Bluetooth device settings. The BCC avoids any application from harming another. The BCC might be an application with a separate API, a native application, or simply a group of settings that are specified by the manufacturer and cannot be modified by the user. BCC is not a class or an interface defined in the specification but an important part of the security architecture. /1/

4.2 Capabilities of JSR 82

This API is intended to provide the following capabilities:

- Manage the local Bluetooth device settings and register services
- Discover Bluetooth devices and services in the neighborhood
- Setup RFCOMM, L2CAP, and OBEX connections between devices
- Using those connections, send and receive data (voice communication not supported)
- Manage and control the communication connections
- Provide security for all the above options /1/

4.2.1 The API Architecture

The aim of the specification was to define an open, non-proprietary standard API that can be used by all J2ME-enabled devices. That’s why it was designed using standard J2ME APIs and CLDC/MIDP's Generic Connection Framework. Some important features: /1/

- The JSR 82 specification addresses the Generic Access Profile, Service Discovery Application Profile, Serial Port Profile, and Generic Object Exchange Profile.
- The specification incorporates the OBEX, L2CAP, and RFCOMM communication protocols in the JSR 82 APIs, primarily because all current Bluetooth profiles are designed to use these communication protocols.
- The Service Discovery protocol is also supported. JSR 82 defines service registration in detail in order to standardize the registration process for the application programmer.
- The specification provides basic support for Bluetooth protocols and profiles. It doesn't include specific APIs for all Bluetooth profiles simply because the number of profiles is growing.

JSR 82 demands that the Bluetooth stack underlying a JSR 82 implementation be qualified for the Service Discovery Application Profile, the Generic Access Profile, and the Serial Port Profile. The stack must also provide access to its Service Discovery Protocol, and to the RFCOMM and L2CAP layers.

Figure 1 shows where the APIs defined in this specification fit in a CLDC/MIDP architecture. The APIs are designed in such a way that developers can use the Java programming language to build new Bluetooth profiles on top of this API as long as the core layer specification does not change. To promote this flexibility and extensibility, the specification is not limited to APIs that implement Bluetooth profiles. JSR 82 includes APIs for OBEX and L2CAP so that future Bluetooth profiles can be implemented in Java, and these are already being used for that purpose.

4.2.2 JSR 82 is Flexible

The JSR 82 APIs can work with both native Bluetooth stacks and Java Bluetooth stacks. In the latter case the APIs call the stack directly; in the former case the APIs go through the virtual machine, which will interface with the native stack. The

minimum configuration is CLDC. Because CLDC is a subset of CDC, applications using these APIs should work on CDC devices. As I already noted, Java developers can expand their options by creating new profiles. JSR 82 standardizes the programming interface. Its two optional packages can be used with any of the J2ME profiles. If the Generic Connection Framework Optional Package for J2SE (JSR 197) is implemented, the JSR 82 APIs should work smoothly with J2SE.

1.3 Pizza Restaurant Application Functional Scenario

- Client Searching

  In this module, the client is searched by the server. The server is active at all times. The client will only be searched if he is in the range of the server. Plus the client should have a Bluetooth-enabled mobile, and after he confirms the request of the server, he can make the benefit of the services that are provided by the restaurant.

- Connection Establishment

  After the client is being detected by the server, the server instantly sends the connection request to the client. After positive acknowledgment from the client side, the connection will be established and then the server and the user will easily be able to communicate with each other and exchange the information.

- Data Transferring

  When the connection is established, the server sends a .jar file to the client which automatically installs the client application on the client's mobile.

  The application contains the restaurant menu, details of the services offered, prices and types of the pizzas. User can then place orders, can see the rate list and after placing the order he also receives the billing information.

- Record Keeping

  All the data that is coming from the client side is saved in the data base that is maintained on the server side.

  There are two types of records.

  I). Client records

  ii). Server records

  There are a number of tables like *order lists, order details, pizzas*, *special offers*, *complaints* and *suggestion*s.
  These tables contain information like lists of the items that are offered by the restaurant, rate lists, different schemes like "*buy one and get one free*, and also complaints and suggestions regarding the food and service offered by the restaurant.
  On the other side, the server sends the .jar file to the client, receives orders from the client and also displays billing information on the user's mobile.

- Dynamic Data Extraction

  When the .jar file executes on mobile, the data that is to be shown on the client side is dynamically obtained from the tables in the database residing on the server side.

4.4 Functional Details

In depth detail of all the functions is explained as follows.

4.4.1 Client Searching

I). Start Server

The server is working at all times. The services of the server are limited to the premises of the restaurant and outside of it, it will not work.

ii). Detection of Bluetooth Devices

When the customer enters the restaurant then the server that is always on and is working will try to detect the Bluetooth of the customer.

4.4.2 Connection Establishment

I). Send Connection Request

When the server detects any Bluetooth-enabled device in its confines, it sends a request to the customer to establish a connection in order to communicate with the client.

ii). Connection to Customer

If customer accepts the connection-request from the server, connection is established. The server gets the mobile ID of the user, and the communication between the server and the client starts.

4.4.3 Data Transferring

I). Upload Jar File

After establishing the connection, the server sends a .jar file to the user, containing the menu, rate list, and special deals offered by the restaurant.

ii). Client-to-Server Data Transfer

When customer gets to see the restaurant details, he can place the order to the server. The “order” contains the information about the food ordered and the table number of the ordering customer.

iii). Billing Information

After enter the order information client side software client can see the billing information and change his order accordingly.

4.4.4 Record Keeping

I). Maintain Database on the Server

All the required data that is to be saved regarding orders of the customers, Suggestions and complaints are kept in the database on the server. This data base includes following tables:

- Order_Details
- Order_List
- Pizzas
- Suggestions_And_Complaints
- Schemes

4.4.5 Dynamic Data extraction

In this part the server takes orders from the database and sees if the customer has updated or cancelled the order. If he has, then the server gets the update on the recent order and makes sure of the accurate delivery.

5 ANALYSIS AND DESIGN DETAILS

Here I present use case diagrams, sequence diagrams, class diagrams , subsystem diagrams and entity relationship diagrams to describe the application. I have client application running on mobile device so I have used Java 2 Micron Edition for client side GUI design and server application running on standalone PC. There I have used Java 2 Standard Edition for server side GUI design.

5.1 Use Case Diagram

There are three actors in this diagram Server Application, Manager, and Client. Manager updates special offers, pizzas and rate list and views database contents like order details. Server application uploads client application and searches for the clients. The client can place orders, view menu and rate list, avails special offers, make suggestions and accepts server connection request.

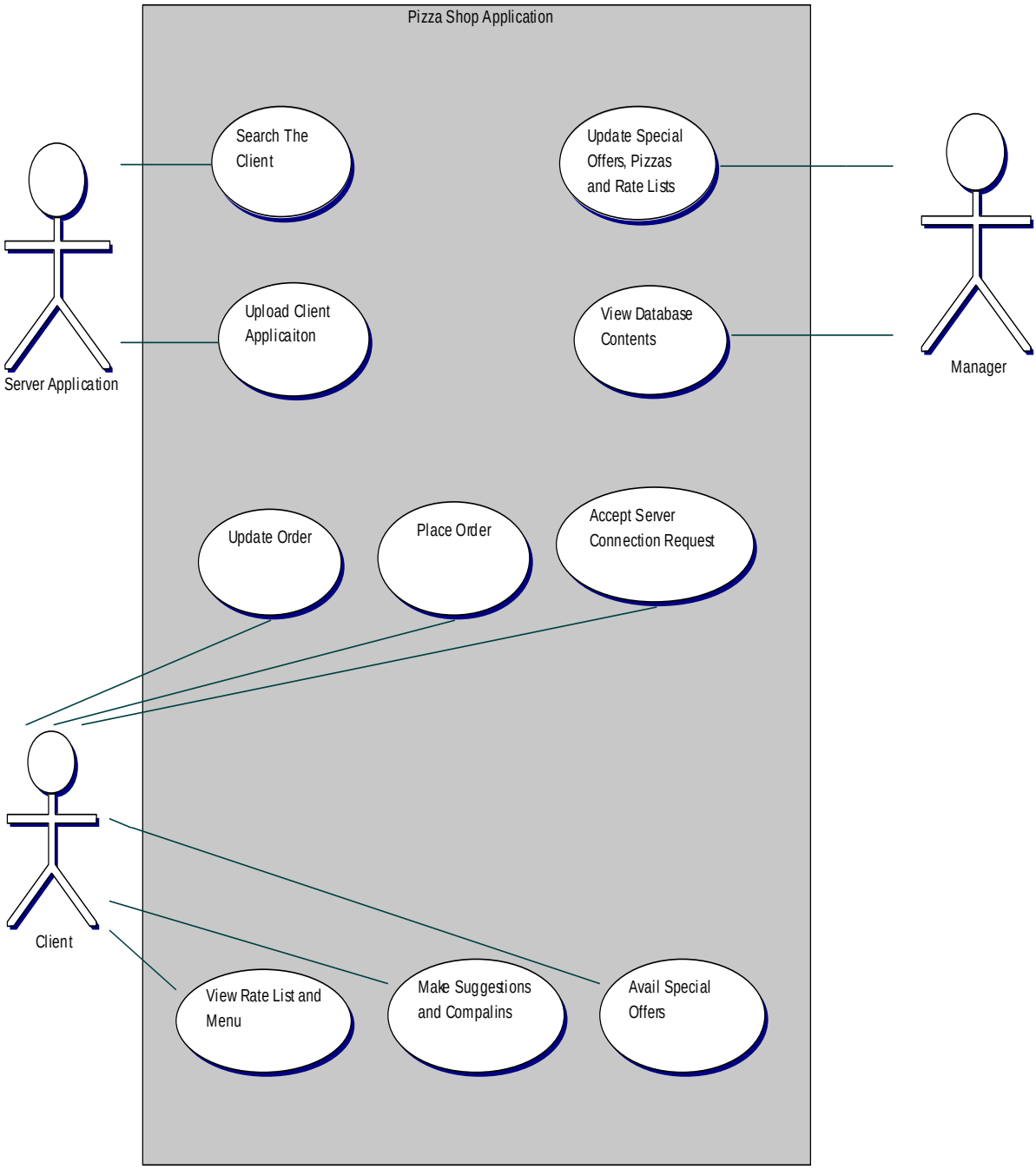


Figure 4. Functions details

5.2 User Classes and Characteristics

In our system there are two categories of classes;

5.2.1 Management

The management class includes the administrator of Pizza Shop Application. Only this class is fully able to use the complete functionality of the system, like viewing system history, making updations, keeping track of orders and so on.

5.2.2 Users

The user will come to the restaurant enclave, get a .jar file from the server through a wireless media, and then, on enabling the application, will be able to see the menu and to place the order.

Management Users

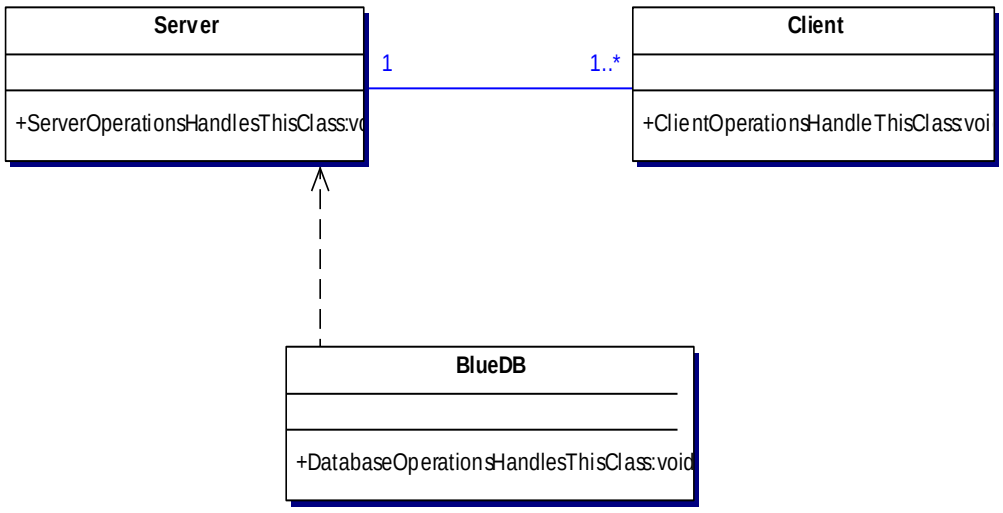


Figure 5. Classes diagram

5.3 Subsystem Partitioning

The whole system is partitioned in to three major parts which are the back bones of the system and they are inter-related with one another.

Client Application

Server Application

Database Connection

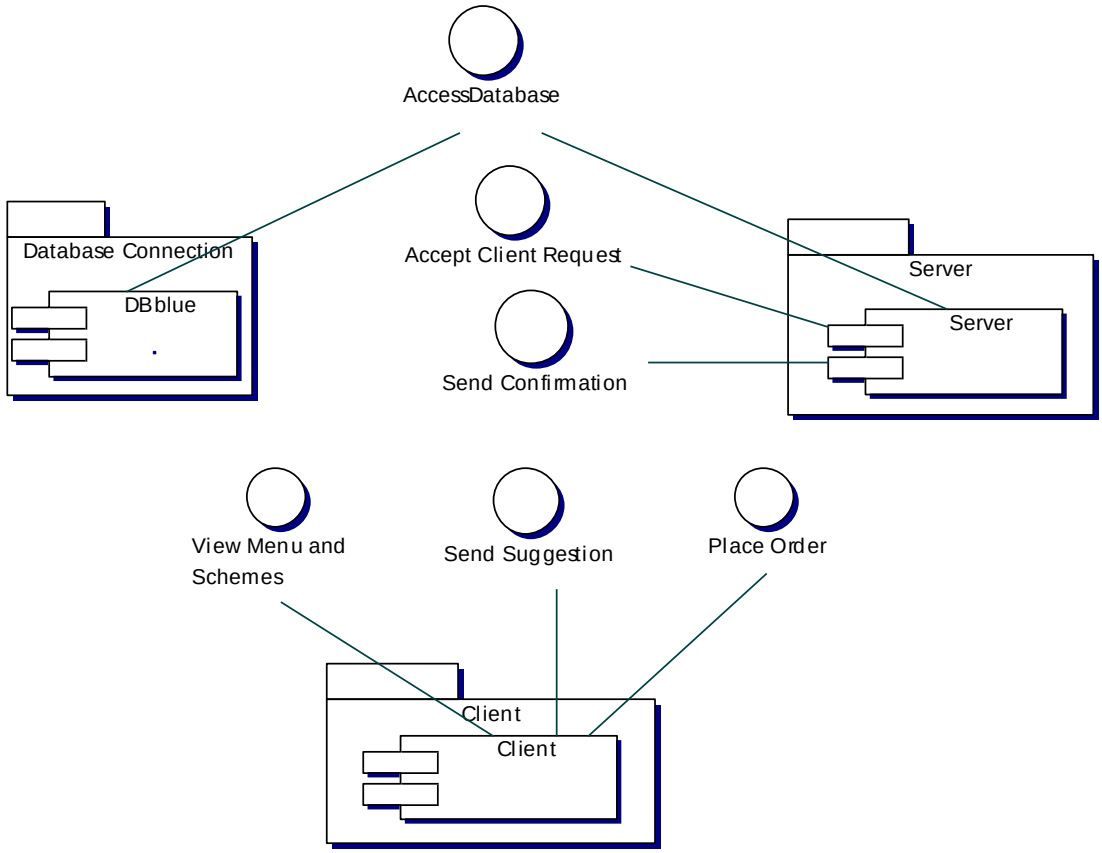


Figure 6. Subsystem Diagram

5.4 Sequence Diagram

The server searches for Bluetooth enabled client, when it finds a device it sends a connection request. When the client approves the request it sends a .jar file to the client. The client opens the application and views schemes, menu and rate list. Through application the client avails schemes, place orders, sends suggestions or complaints and sees help option. The client receives order confirmations and gets billing information.

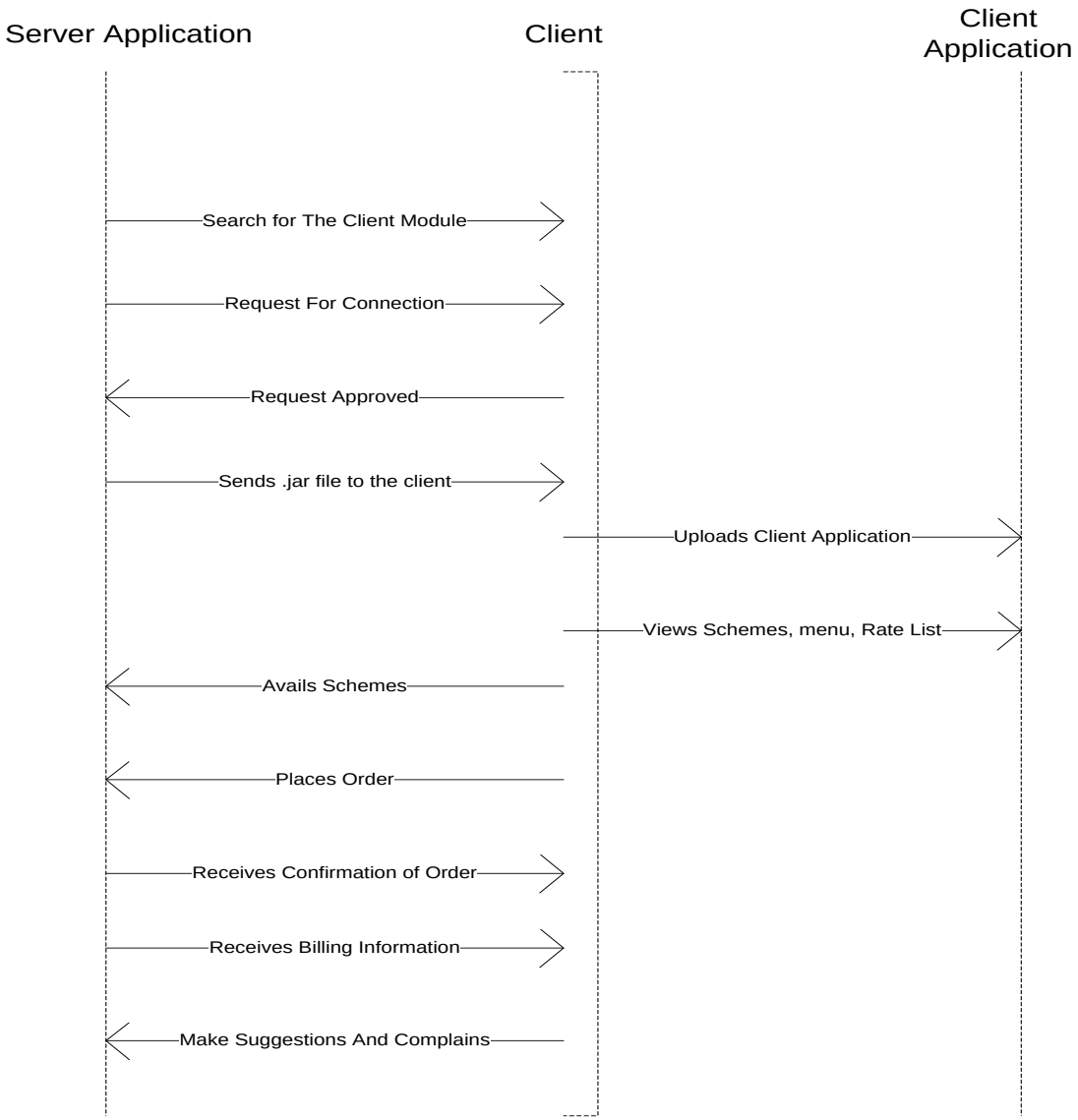


Figure 7. Sequence Diagram

5.5 Entity Relationship Diagram

In this diagram there are 5 tables *order lists, order details, pizzas, schemes, complains.* Order list contains general information of order like customer name, order date and customer's phone number. Order details table depends on orderlist table for order id and has pizzas information and billing information.

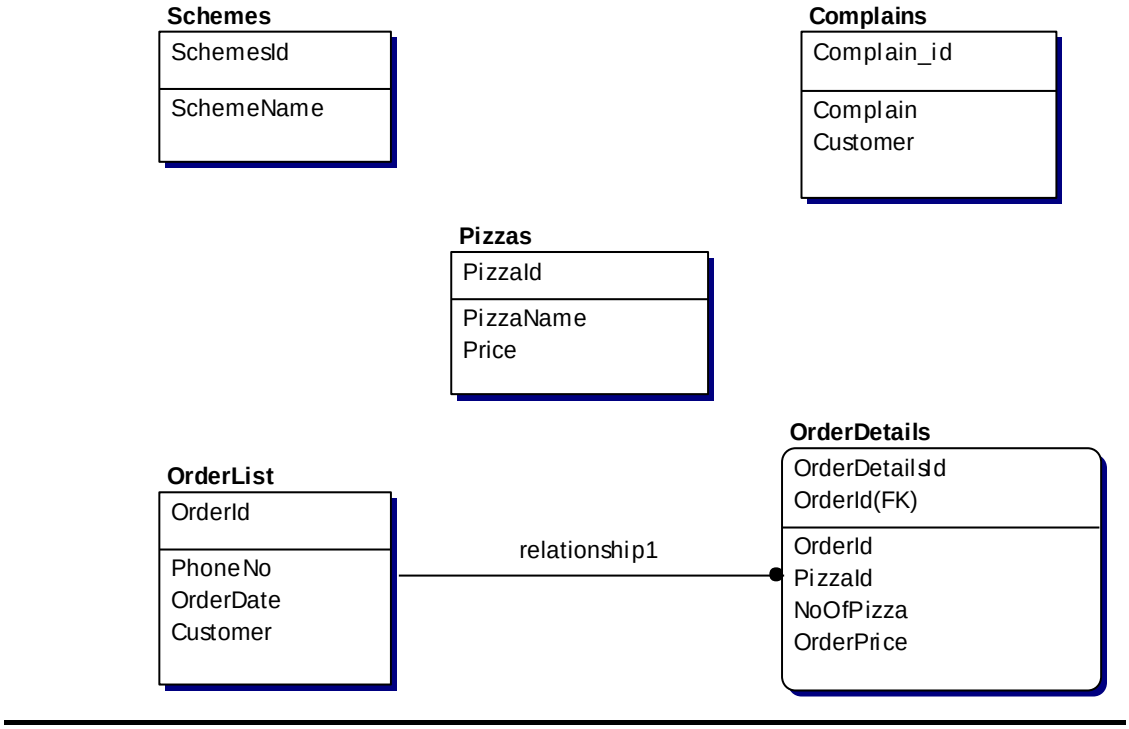


Figure 8. Entity-Relationship Diagram

5.6 Client Graphical User Interfaces

The screenshots/interfaces available to the users are given below:
In following figure the client has opened our software in his mobile device and we can see his Bluetooth device is on. The client can choose to exit from our application

or choose device search option to search pizza server. In options command he can choose to display Main Menu, if he is outside the range of pizza restaurant he can see main menu but cannot place order from there before connecting to server which is done by device search command.

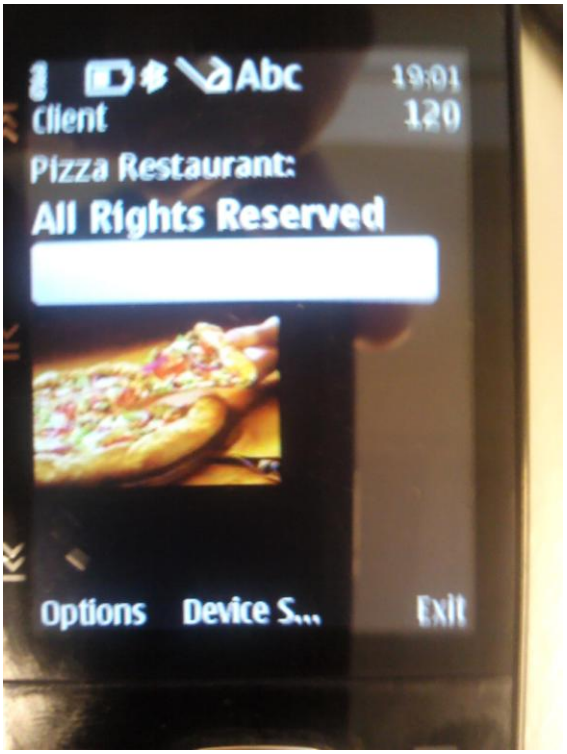


Figure 9. Main Screen

In the following picture we can see the client searching for Pizza Restaurant Server. The message “Searching Pizza Restaurant Server” displays after selecting Device Search command.

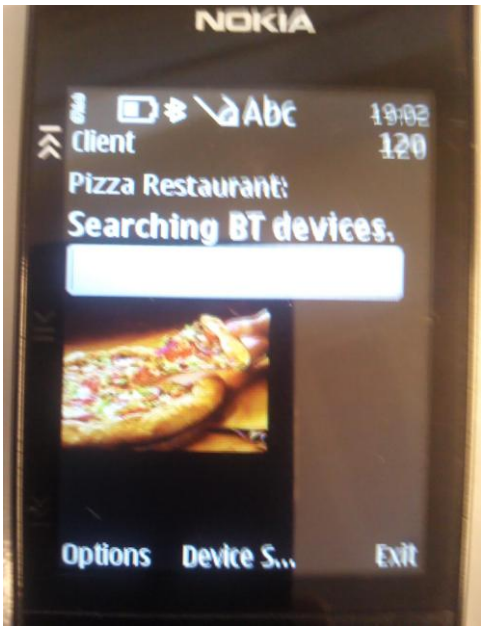


Figure 10. Server Search

In the following picture we can see that the client is connected to the server. He can see the menu to place order, check scheme, give feedback and see help.

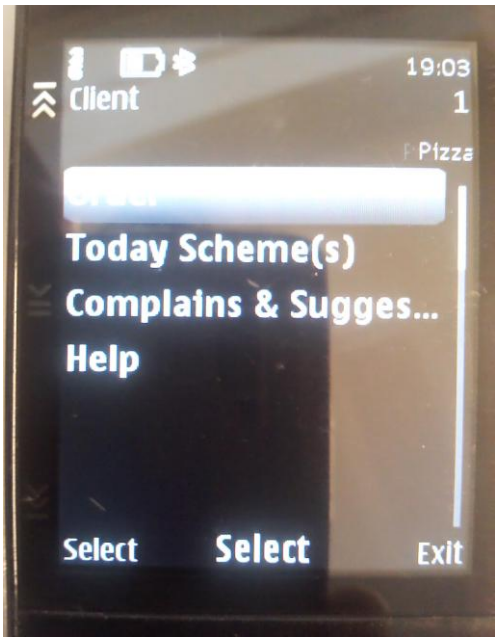


Figure 11. Menu

In the following figure the client has chosen to place order and after that he can see pizzas list and rate list to select for order.

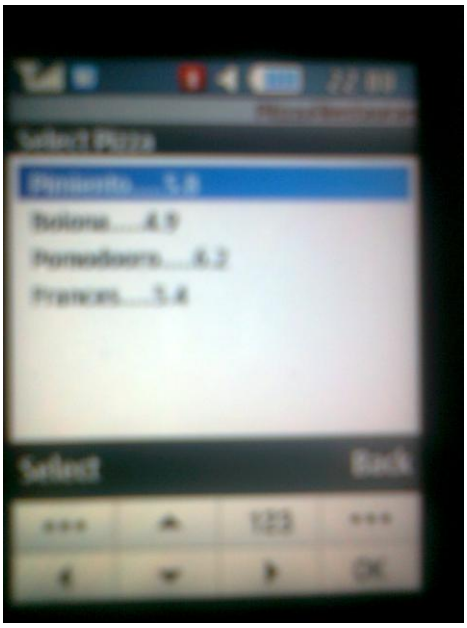


Figure 12. Pizzas List

In the following figure the client is required to enter order detail, his name, phone number, no of pizzas he likes to order and size of pizza.

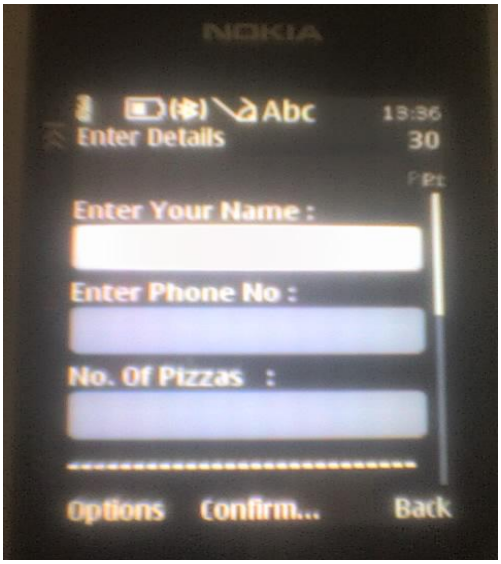


Figure 13. Entering Order Detail

In the following figure the client can see billing information after entering order details. The total cost is calculated from runtime by the software.

Figure 14. Billing Information

In this figure the client has selected 2nd command show scheme and the following figure shows him today's discount offer from database server.

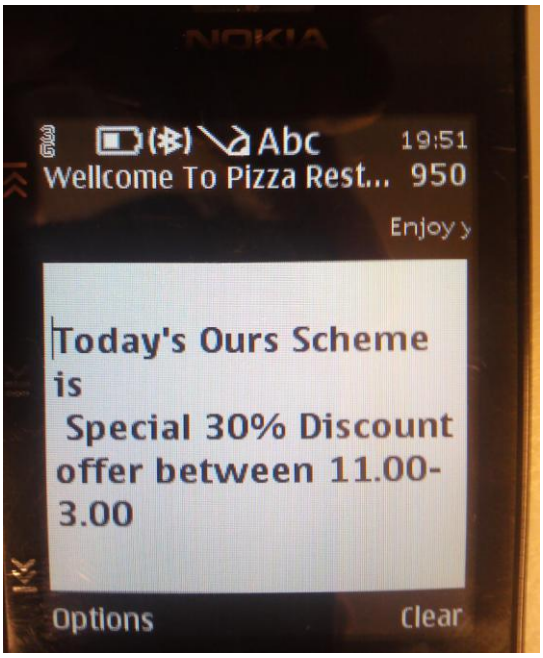


Figure 15. Special Offers

5.7 Server Graphical User Interfaces

This is the server's side view of the application. This figure shows the first screen which appears after running pizza restaurant server. There are two buttons; start server and send application to client, both these perform different functions which are shown in the next figures.

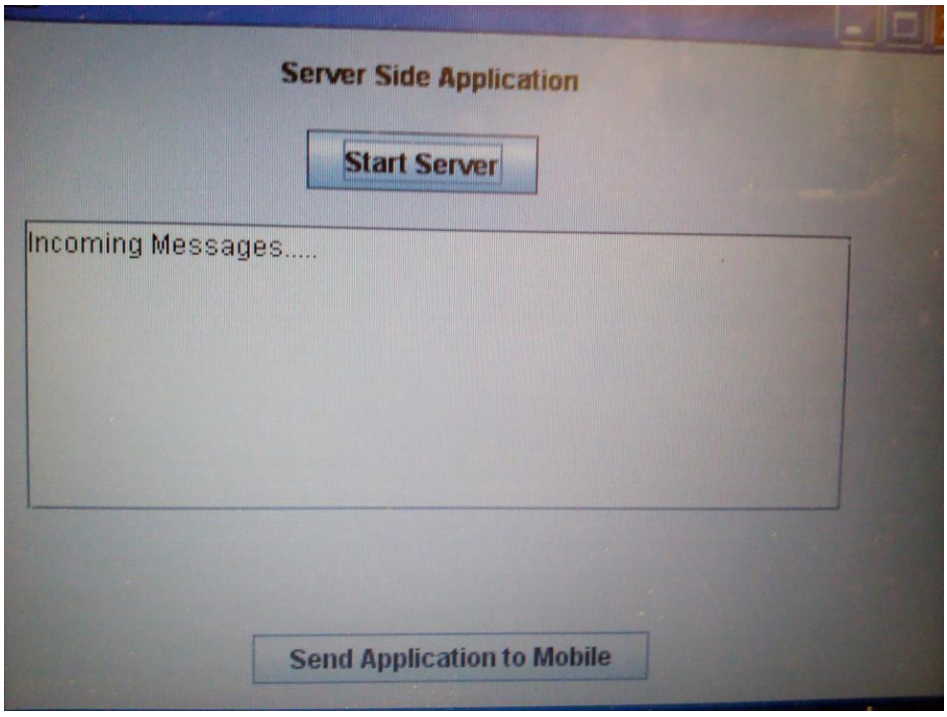


Figure 16. First Screen

This figure is shown when the manager presses send application button on the mobile. The following screen shows that the server is searching for Bluetooth devices in the vicinity. When a device is found, it tries to find service on the device to send application. When a service is found, the server sends application file "Final.jar", which automatically installs the client's mobile device, and after receiving the application the client can start using application at the same time.

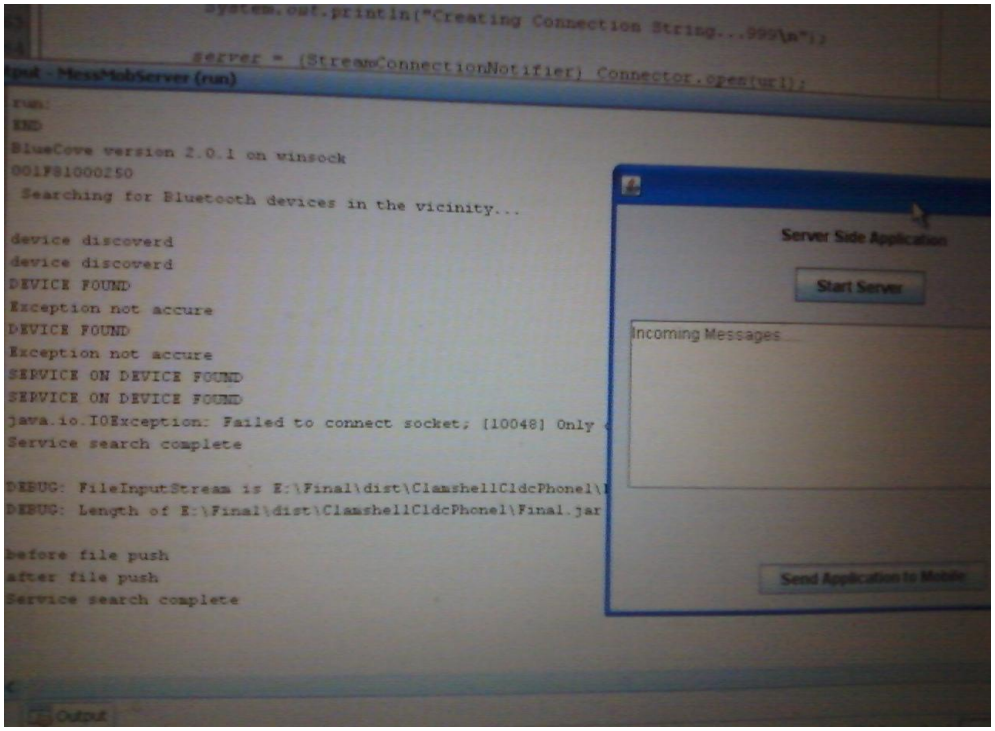


Figure 17. Sending Application to Client

The next figure displays the result of pressing Start Server button. We can see the command prompt display that server is running and waiting for a client. Then a client named Saied connects to the server and he places an order which is shown here at command prompt level and in the next coming figure in pop up display.

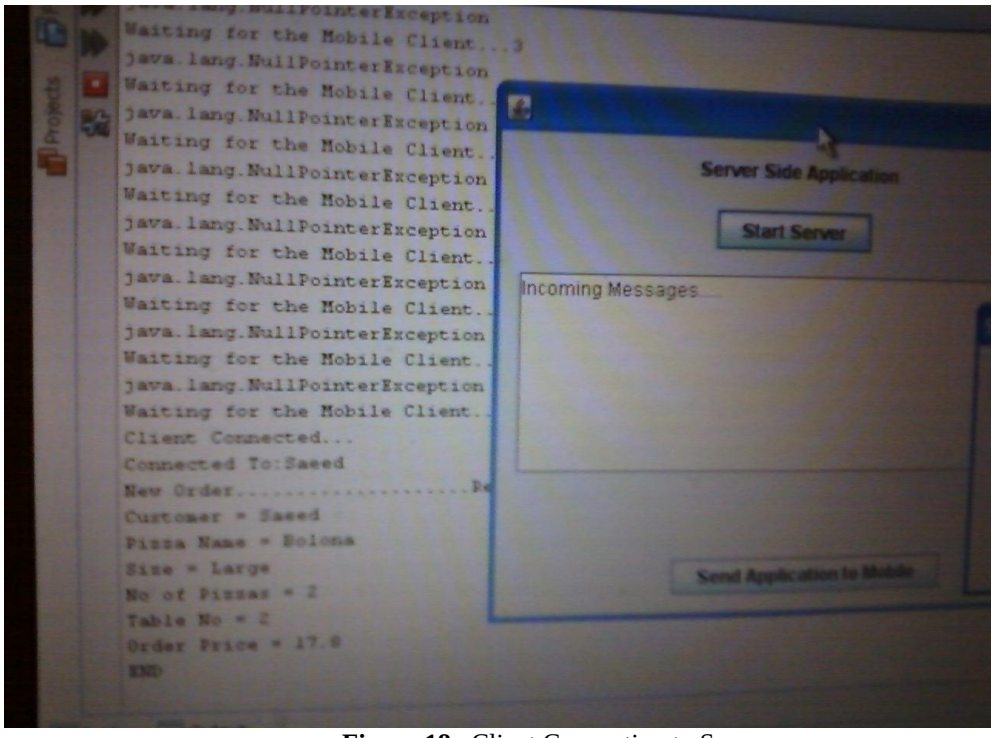


Figure 18. Client Connecting to Server

The next figure shows an order coming from the client's side. A pop up window appears showing the order at server.

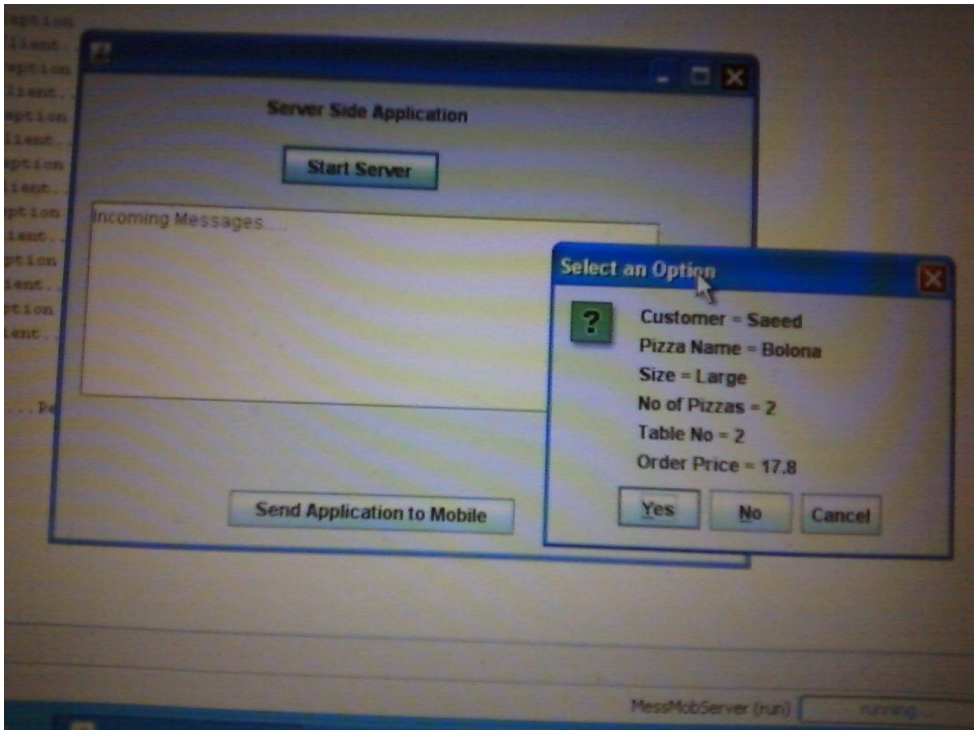


Figure 19. Order Display Window

6 IMPLEMENTATION MODULE

Implementation of Bluetooth programming is explained as follows.

6.1 Packages

The Java APIs for Bluetooth consists of two packages that depend on the CLDC `javax.microedition.io` package:

- `javax.Bluetooth`: core Bluetooth API
- `javax.obex`: APIs for the Object Exchange (OBEX) protocol

Each of the above packages signifies a separate optional package, which means that a CLDC implementation can include either package or both. MIDP-enabled devices are expected to be the kind of devices to fit in this specification. Again, the OBEX APIs are defined independently of the Bluetooth transport layer and packaged separately. /1/

6.2 Application Programming

The anatomy of a Bluetooth application has five parts: /1/

- Stack initialization
- Device management
- Device discovery
- Service discovery
- Communication.

Now we explain one by one that how basic communication is done between Server PC and Client (mobile)

6.2.1 Stack Initialization

The Bluetooth stack is the piece of software that controls your Bluetooth device, so you need to initialize the Bluetooth stack before doing anything else. The initialization process contains a number of steps whose purpose is to get the device ready for wireless communication. Unfortunately, the Bluetooth specification leaves implementation of the BCC to vendors, and different vendors handle stack initialization differently. On one device, it may be a series of settings that cannot be modified by the user, and on another it may be an application with a GUI interface. For example, Atinav's Java Bluetooth solution requires the developer to initialize the stack with a series of settings like the ones in the following code fragment - note well that the APIs invoked are not part of JSR 82: /1/

For example:

```
// seting the port number
BCC.setPortNumber("COM1");
// seting the baud rate
BCC.setBaudRate(50000);
// set the connectable mode
BCC.setConnectable(true);
// set the discovery mode to General Inquiry Access Code
BCC.setDiscoverable(DiscoveryAgent.GIAC);
...
```

6.2.2 Device Management

LocalDevice and RemoteDevice are the two main classes in the Bluetooth APIs, which provide the device-management capabilities defined in the Generic Access Profile. `LocalDevice` depends on the `javax.Bluetooth.DeviceClass` class to retrieve the device's type and the kinds of services it offers. The `RemoteDevice` class represents a remote device (a device within a range of reach) and provides methods to retrieve information about the device, including its Bluetooth name and address. The following code fragment retrieves that information for the local device: /1/

```
...
// get the local Bluetooth device object
LocalDevice local = LocalDevice.getLocalDevice();
// get the Bluetooth address of the local device
String address = local.getBluetoothAddress();
// get the name of the local Bluetooth device
String name = local.getFriendlyName();
...
```

And following code fragment retrieves the information about a remote device:

```
...
// get the device that is at the other end of
// the Bluetooth OBEX over RFCOMM connection,
// L2CAP connection, or Serial Port Profile connection
RemoteDevice remote =
    RemoteDevice.getRemoteDevice(
        javax.microedition.io.Connection c);
// get the Bluetooth address of the remote device
String remoteAddress = remote.getBluetoothAddress();
// get the name of the remote Bluetooth device
String remoteName = local.getFriendlyName(true);
...
```

The `RemoteDevice` class also provides methods to authorize, authenticate, or encrypt data transferred between local and remote devices.

6.2.3 Device Discovery

Your Bluetooth device has no idea of what kind of other Bluetooth devices are in the area. Perhaps there are mobile phones, laptops, printers, desktops, or PDAs in the area. In order to find out the core Bluetooth API's `DiscoveryAgent` class and `DiscoveryListener` interface provide the necessary discovery services.

A Bluetooth device can use a `DiscoveryAgent` object to obtain a list of accessible devices, in any of three ways:

The `DiscoveryAgent.startInquiry` method puts the device into an inquiry mode. To take advantage of this mode, the application should specify an event listener that will respond to inquiry-related events. `DiscoveryListener.deviceDiscovered` is called each time an inquiry finds a device. When the inquiry is completed or canceled, `DiscoveryListener.inquiryCompleted` is invoked.

If the Bluetooth device doesn't want to wait for devices to be discovered, it can use the `DiscoveryAgent.retrieveDevices` method to retrieve an existing list. Depending on the parameter passed, this method will return either a list of devices that were found in a previous inquiry (*cached*), or a list of *pre-known devices* that the local device has told the Bluetooth Control Center it will contact often. /1/

These three code snippets demonstrate the various approaches:

```
...
// retrieve the discovery agent
DiscoveryAgent agent = local.getDiscoveryAgent();
// put the device in inquiry mode
boolean complete = agent.startInquiry();
...
```

```
...
// get the discovery agent
DiscoveryAgent agent = local.getDiscoveryAgent();
// return a list of devices found in a previous inquiry
RemoteDevice[] devices =
    agent.retrieveDevices(DiscoveryAgent.CACHED);
...
```

```
...
// get the discovery agent
DiscoveryAgent agent = local.getDiscoveryAgent();
// return a list of pre-known devices
RemoteDevice[] devices =
```

```
        agent.retrieveDevices(DiscoveryAgent.PREKNOWN);
    ...
```

6.2.4 Service Discovery

Once the local device has discovered at least one remote device, it can begin to search for available *services*. The searchServices() method of the DiscoveryAgent class lets you to search for services on the RemoteDevice. When services are found, the servicesDiscovered() method will be called by the JVM if your object implemented the DiscoveryListener interface. This callback method also passes in a ServiceRecord object that pertains to the service for which you searched. With a ServiceRecord in hand, you can do plenty of things, but you would most likely would want to connect to the RemoteDevice where this ServiceRecord originated. Note that the API provides mechanisms to search for services on remote devices, but not for services on the local device. Service Discovery is just like Device Discovery in the sense that you use the DiscoveryAgent to do the discovering. /1/

6.2.5 Service Registration

Before a service can be discovered by a Bluetooth device, it must first be *registered* in the Service Discovery database (SDDB) and advertised on a *Bluetooth server device*. The server is responsible for:

- Creating a service record that represents the service offered.
- Adding the service record to the server's SDDB, so it's visible and available to clients.
- Registering the Bluetooth security measures related with the service (enforced for connections with clients).
- Accepting connections from potential clients.
- Modifying the service record in the SDDB if the service's attributes change.

- Deleting or disabling the service record in the SDDB when the service is no longer available.

The following annotated code fragment gives you a flavor of the effort involved in registering a service using the Java APIs for Bluetooth: /1/

1. To create a new service record that describes the service, invoke `Connector.open` with a server connection URL argument, and cast the result to a `StreamConnectionNotifier` that represents the service:

```
...
StreamConnectionNotifier service =
    (StreamConnectionNotifier) Connector.open("someURL");
```

2. To acquire the service record created by the server device:

```
ServiceRecord sr = local.getRecord(service);
```

3. This indicates that the service is ready to accept a client connection:

```
StreamConnection connection =
    (StreamConnection) service.acceptAndOpen();
```

   Note that `acceptAndOpen` blocks until a client connects.

4. When the server is ready to exit, this closes the connection and removes the service record:

```
service.close();
...
```

6.2.6 Communication

To use a service on a remote device for a local device, the two devices should share a common communications protocol. For applications to access a wide variety of Bluetooth services, the Java APIs for Bluetooth gives mechanisms that allow connections to any service that uses RFCOMM, L2CAP, or OBEX as its protocol. The application *can* access the service, if a service uses another protocol (such as TCP/IP) layered above one of these protocols, but only if it implements the supplementary protocol in the application, using the CLDC Generic Connection Framework.

JSR 82 implements the OBEX API (`javax.obex`) independently of the core Bluetooth API (`javax.Bluetooth`), because the OBEX protocol can be used over several different transmission media - wired, Bluetooth radio, infrared and others. OBEX API is an optional package which can be used either with the core Bluetooth package or independently. /1/

6.2.7 Serial Port Profile

The RFCOMM protocol, layered over the L2CAP protocol, emulates an RS-232 serial connection. The Serial Port Profile (SPP) can make communication easy between Bluetooth devices by providing a stream-based interface to the RFCOMM protocol. Some capacities and constraints to note:

- A single Bluetooth device can have at most 30 active RFCOMM services.
- A device can support only one client connection to any given service at a time.
- Two devices can share only one RFCOMM session at a time.
- Up to 60 logical serial connections can be multiplexed over this session.

For a server and client to communicate using the Serial Port Profile, each must perform a few simple steps.

As the following code fragment demonstrates, the server must:

1. Construct a URL that indicates how to connect to the service, and store it in the service record
2. Make the service record available to the client
3. Accept a connection from the client
4. Send and receive data to and from the client

The URL placed in the service record may look something like:

```
btspp://102030405060740A0B0C0D0E100:4
```

This line of code tells that a client should use the Bluetooth Serial Port Profile to make a connection to this service, which is identified with server channel 4 on a device whose address is 102030405060740A0B0C0D0E100. /1/

```
...
// assuming the service UID has been retrieved
String PizzaServiceURL =
    "btspp://localhost:"+serviceUID.toString());
// more precisely:
String PizzaServiceURL =
    "btspp://localhost:10203040607040A0B0C0D0E100;name=Pizza
        Server";
try {
    //to create a server connection
    StreamConnectionNotifier notifier =
   (StreamConnectionNotifier) Connector.open(PizzaServiceURL);
    //to accept Bluetooth client connections
    StreamConnection connection = notifier.acceptAndOpen();
    //these lines of code are to prepare to send/receive data
    byte text[] = new byte[100];
    String tmp = "hello there, client";
    InputStream fromclient= connection.openInputStream();
```

```
        OutputStream toclient = connection.openOutputStream();
        //to send data to the Bluetooth client
        toclient.write(tmp.getBytes());
        //to read data from Bluetooth client
        fromclient.read(text);
        connection.close();
    } catch(IOException e) {
      e.printStackTrace();
    }
    ...
```

6.3 Hardware Interfaces

The following are some of the hardware requirements, which are necessary for our system.

Processor:

- Pentium 4 with 1.6GHz or greater
- The processor of good quality like Intel is recommended.

Memory:

- RAM: 512 MB (at least 1 GB recommended)
- Hardware with 128 MB of texture memory (256MB recommended)

Hard Disk Space:

- At least 80GB (approx.) system hard disk space is required for database and users accounts. So at least 80GB System hard disk space is recommended for the system to work efficiently.

Mobile Unit:

- The application is designed to run on Java-enabled mobile phones. Bluetooth provision is a compulsion.
- The current version of Pizza Shop Application is intended for only Nokia 60 Series.

6.4 Software Interfaces

The following are some of the software requirements, which are necessary for our system.

Operating System:

- The Windows based operating system including Windows 2000 Professional, Windows NT and Windows XP, Window Vista, Window 7.0 can be used for our system.

Programming Platform:

- The java language will be used as programming language. The NetBeans IDE is used as the development environment. Java is used because it is open source, powerful, relatively simpler and it is productive.

Database:

Database will be in MYSQL, because it is more easy and understandable.

Toolkit:

Java Platform Micro Edition Software Development Kit 3.0
Java ME Platform SDK is a state-of-the-art toolbox for developing mobile applications. It incorporates CLDC, CDC and Blu-ray Disc Java (BD-J) technology into one SDK. Java ME SDK 3.0 is the successor to the popular Java Wireless Toolkit 2.5.2 and Java Toolkit 1.0 for CDC. It provides a standalone development environment, device emulation and a set of utilities for rapid development of Java ME applications. /1/

7 CONCLUSIONS

The most challenging part of the thesis was concerning internal details of Bluetooth implementation part to make it functional and make the connection between the two applications on two different devices. Many low level details which give error signals when working within the actual scenario. There is a limitation in this application when I try to send the application to all known devices then the Bluetooth API tries to use the same port number for all opened connections for sending the application. How the port number for any new opened connection can be changed remains unsolved.

The Bluetooth API sends the application to the first mobile but for every 2nd mobile an error message "Fail to connect to socket: [10048] only one connection allowed" is shown. I have checked this application on Nokia and Samsung mobiles and I have shown pictures for both mobiles that it is working. Instead of the Nokia mobile shown in pictures, this application also functions successfully on Nokia mobile classic 6700. However, there is a need to do more for all kind of mobiles for this application to work with. Nokia S60 series is supported in this version and only Samsung GT-S3370E is checked and it is working.

References

/1/ The Java APIs for Bluetooth Wireless Technology, .[accessed 10.6.2010] .Available in www-form:

< URL: http://developers.sun.com/mobility/midp/articles/Bluetooth2/>

/2/ Connection to MySQL database Using Net beans, .[accessed 15.2.2011] .Available in www-form:

< URL: http://www.netbeans.org/kb/60/ide/mysql.html>

/3/ Ghodrat Moghadampour , Wireless Java Programming , .[accessed 5.2.2010] .Available on intranet

<URL: https://portal.puv.fi/course/view.php?id=699>