

# **Mobile Applications with J2ME and JSR82: Bluetooth enabled Java Applications for Mobile Phones**

**Student: Borja Gomez Zarceño**  
**Mentor: Dr. Stephan Rupp**

**Institute of Communication Networks and Computer  
Engineering**

## ***Abstract***

Bluetooth came in so as to make the use of home electronics more friendly to users. It permits any kind of electronics equipment to establish their own connections, without cabling and without direct actions from the user. Java language programming and platform appear to join Bluetooth technology to be the base of next generation wireless applications. Developers have built a platform providing a run time environment for embedded systems upon small devices. This paper gives an overview of Bluetooth and introduces the Java platform for developing Mobile Applications. This Java platform (J2ME) and the Java Application Programmer Interface for bluetooth (JSR 82) will be explained.

## **1. Bluetooth**

There is a variety of ways to connect electronic devices: a cable connecting a processing unit to a display, a data cable connecting a mobile telephone to a PC, radio waves connecting a cordless telephone to a base unit, infrareds do with a remote control to a TV.

Bluetooth appears to be a different way to build up connections between devices placed in proximity. This radio frequency technology can be thought as a cable replacement technology that will not only replace cables, but be the base to develop next generation wireless applications, which will be built upon this technology. It offers a variety of other services, apart from connecting devices, and it creates opportunities for new usage models.



Figure 1.1 Bluetooth connections

Among the bluetooth characteristics some of them are remarkable: bluetooth is wireless; automatic; cheap; deals with data and voice; signals are omnidirectional and a device can send to multiple devices at the same time; signals can go through walls; bluetooth is secure, it uses frequency hopping what makes it difficult for the communication data to be intercept; it is standardized, governments have arranged a standard about the ISM band so that it is possible to use devices wherever.

Bluetooth technology is to be used for the next applications: *file transfer*, the content of one mobile phone can be dragged into another one; *ad-hoc networking*, network spontaneously form networks that persist as long as they are needed; *device synchronization*, data handled in one device is updated in all devices; hands-free packages for accessing the telephone service while driving; *peripheral connectivity*; *mobile payments*, a bluetooth enabled mobile phone can communicate with a bluetooth enabled machine to buy something and pay for it.

## **1.1 Topology**

Bluetooth devices are grouped in piconets. Each piconet consist of a single master and different slaves. A master and a slave build a point-to-point communication. If there are more than one slaves, a point-to-multipoint connection is established. The master is the unit which initiates the communication. A device in one piconet can connect to a device belonging to another piconet, forming a scatternet. As we can see in the figure, in a scatternet, a master in one piconet can be a slaved in other piconet.

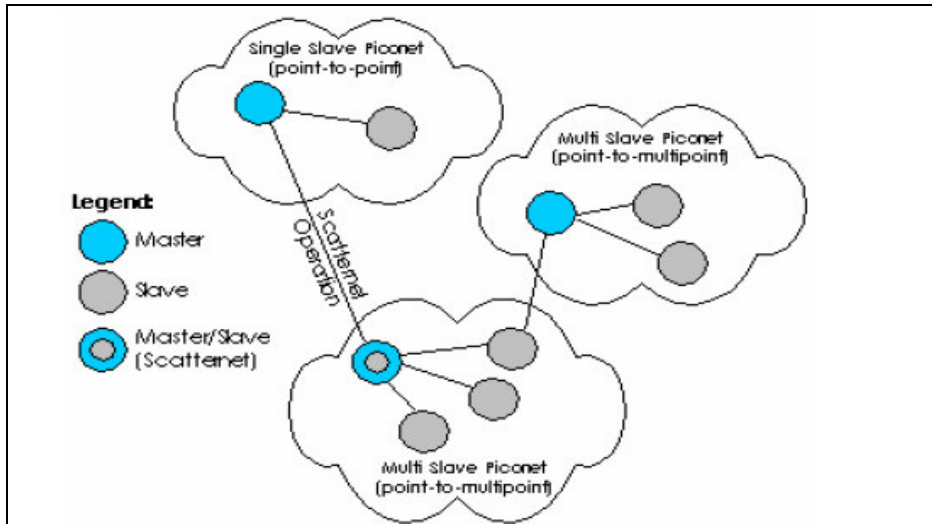


Figure 1.2 Bluetooth Network Topology

Bluetooth uses a time division multiplex scheme for transmission, dividing time into slots. To support full-duplex transmission, the master device uses an even-numbered slot, while slaves use an odd-numbered slot.

## 1.2 Protocol Stack

The bluetooth specification ensures that all the devices supporting this technology are able to communicate with each other no matter the part of the world where they are. A well defined protocol stack is then defined. A layer level view of the architecture of the Bluetooth technology is shown as follows

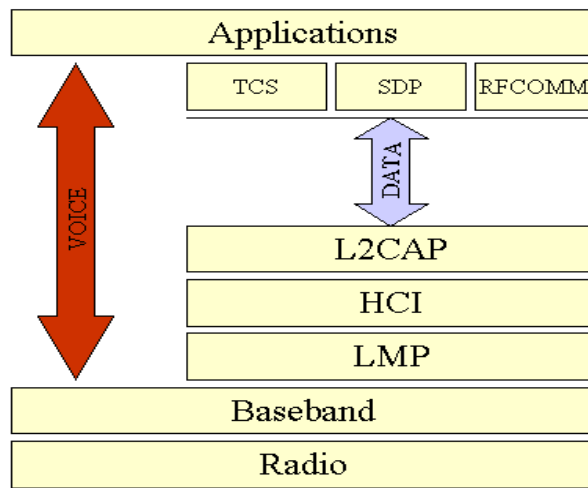


Figure 1.3 Protocol Stack

### **1.2.1 The Radio Layer**

The radio layer is the physical wireless connection. The modulation is based on fast frequency hopping, allowing to avoid interferences with other devices using the same band. The frequency band from 2.402 GHz to 2.480 GHz (ISM band) is divided into 79 channels, each one 1MHz of bandwidth. The transmission frequency hops from one channel to another about 1600 times per second.

### **1.2.2 The Baseband Layer**

The baseband layer controls and send data packets through the radio link. It does provide a transmission channel for data as well as for voice. This layer provides two types of links: A *Synchronous Connection Oriented* (SCO) link for voice and an *Asynchorous Connectionless Link* (ACL) only for data . To ensure data integrity, when there has been a failure in a packet transmission, ACL support retransmission. In SCO there is no retransmission of packets.

SCO links are point to point symmetric connections, where time slots are reserved for each direction of transmission. The master transmits in one time slot, whereas the slave is allowed to answer back in the next immediate time slot. At most, there can be, for a master, three SCO links at the same time either to a single or to multiple slaves. On the other hand, in ACL links there is no reservation of time slots. These links se the time slots that are not being used by SCO connections. These links support point to multipoint transmissions, in which the master may address specifically a slave or not (broadcast messages). In the former case, only slaves addressed may respond during the next time slot.

### **1.2.3 The LMP, HCI and L2CAP Protocols**

The Link Manager Protocol (LMP) makes use of the services from the layer below, that is, from the links set up by the baseband layer to establish connections and to manage the piconets.

The Host Controller Interface is the layer providing an interface between software and hardware functions. From there to the top of the protocol stack, functions are implemented by software. It is a driver interface for the physical bus connecting these components. Sometimes this layer is not required.

The Logical Link Control and Adaptation Protocol (L2CAP) negotiates QoS. As well, this layer receives application data and changes it to Bluetooth format. It provides connection-less and connection-oriented services to upper layers. This is done with multiplexing capabilities in order to differentiate the protocols above: SDP, RFCOMM and TCS.

### **1.2.4 RFCOMM, SDP and TCS protocols**

The RFCOMM protocols is a transport protocol emulating a RS232 serial ports. It permits up to 60 different connections between bluetooth enabled devices.

The Service Discovery Protocol allows applications running over bluetooth to find out services and service characteristics registered in other bluetooth devices. The TCS define the interface needed to connect and disconnect a call, including signalling the devices to participate in the connection.

### **1.3 Bluetooth profiles**

To assure interoperabilty among bluetooth devices, provided by different vendors and manufacturers, a bluetooth device must conform to a particular profile, which defines capabilities and roles for applications, defining mandatory options and parameters for each protocol of the stack. A profile can be seen then as a vertical sight of the protocol stack. Some of these profiles are slightly described as follows

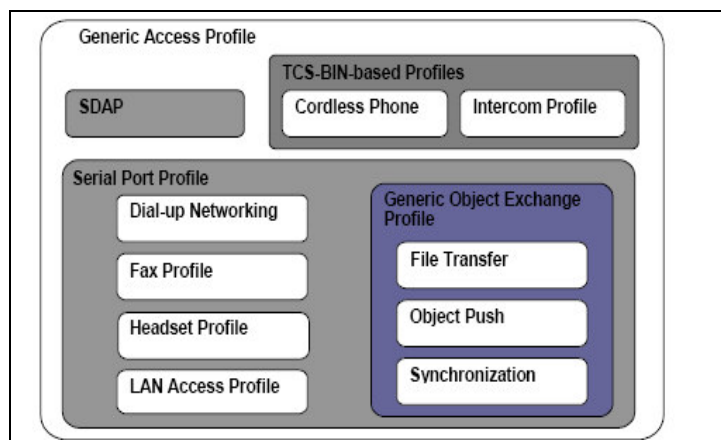


Figure 1.4 Bluetooth Profiles

1. The Generic Access Profile (GAP): for defining connection procedures, as well as link management and device discovery. That is, for describing the use of the lower layers of the protocol stack (LC and LMP).
2. The Service Discovery Application and Profile (SDAP): for defining characteristics in order to discover services around and to bring available information about these services. It defines protocols and procedures to be used by discovery applications in devices using the service discovery protocol.

3. The Serial Port Profile (SPP) for establishing connections.
4. The LAN access profile: for defining how to access a LAN and PPP procedures.

## **2. J2ME**

The Java2 MicroEdition platform joins the Bluetooth technology to be one of the most exciting offerings in the wireless industry. J2ME was defined by Sun Microsystems to meet the new needs of Java developers working on consumer and small embedded systems. J2ME itself is not a specification, but a group of them defining how Java technology is upon devices with few resources compared to a PC. This platform is portable, so applications follow the Java philosophy “once written run anywhere”. It appears as a tool to let us write custom applications and run them on mobile Bluetooth enabled devices.

Devices using this Java Platform are: PDAs, cell phones, television set top boxes, remote telemetry units... and other embedded devices. These are heterogeneous devices regarding processor power, memory, persistent storage and user interface. It is difficult to provide an optimal functionality for all these embedded devices due to this heterogeneity. There was a first division of the devices into two sections considering the resources above mentioned, but not taking into account the function or use of the device. For the latter reason, there was a subdivision into classes of devices.

There are two J2ME configurations corresponding to the former division above mentioned: Connected Device Configuration (CDC) and Connected Device Limited Configuration (CDLC), each one providing a JVM (Java virtual machine) and different libraries and APIs in order to create a run-time programming environment for a group of devices. Configurations provide a common denominator subset of Java suitable for the characteristics of devices to which they are intended.

Within a configuration, there is still much heterogeneity in aspects such as user interface, function and usage. Configurations do not specify the user interface toolkit and persistent storage APIs. This is the task of profiles.

A profile is a set of JAVA APIs for a particular class of device, considering its function, such as mobile phones. It is built over the platform provided by the correspondent configuration. It is meant to couple this configuration. The Foundation Profile and the Mobile Information Device Profile (MIDP), correspond to CDC and CDLC respectively. For example, MIDP 2.0 joins CLDC to provide a run-time environment.

J2ME defines a small core API defines a core API to be implemented in every J2ME compatible device, deployed in different configurations. The next sets fo Java classes are to be in every configuration: java.util and java.lang

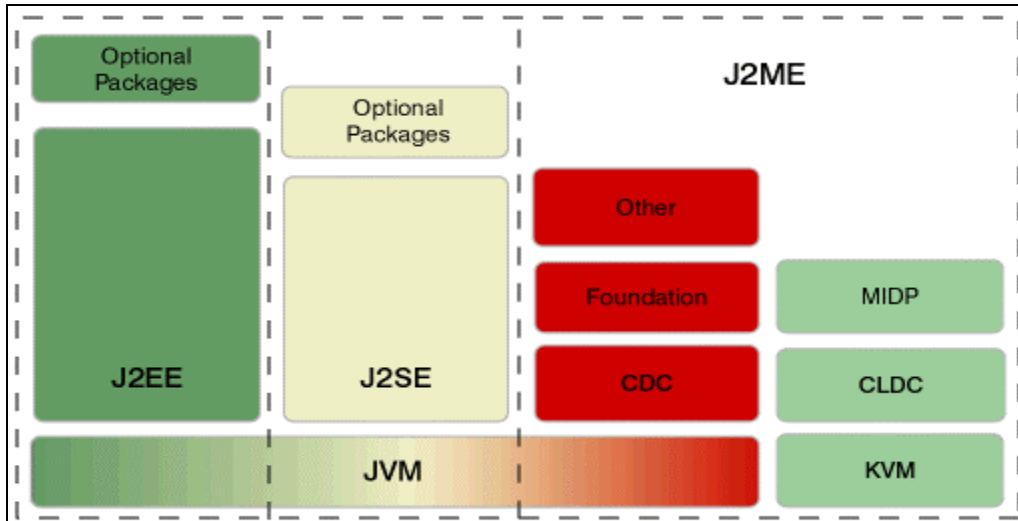


Figure 2.1 Organization of the J2ME Platform

Apart from profiles and configurations, there are optional packages. An optional package is a set of APIs, technology-specific extending the functionality of the application environment.

The KVM is a J2ME virtual machine intended for resource constrained devices. It is separate from the Java Virtual Machine. In this virtual environment the CLDC configuration is meant to give service to devices having 512 KB or less memory, together with a limited power supply, limited connectivity to networks and simple user interface. CLDC is supposed to create applications for devices such as mobile phones, PDAs... It will provide a set of libraries and APIs. That is, classes and methods. CLDC will be later extended by profiles (MIDP) and optional packages to form new libraries and APIs, making use of the inheritance concept in Java language, allowing us to make use of what is already defined in CLDC to address it to specific applications for specific devices.

The Mobile Interface Device Profile abstracts the various physical features of a mobile device into a set of Java classes. The MIDP standard is supported by most mobile manufacturers. Applications upon MIDP are called MIDlets.

JSR 82 extends all this functionality from CLDC and MIDP to provide an API for bluetooth enabled devices. This specification appears to define the architecture and the APIs to enable an open application run time environment.

### **3. Java APIs for bluetooth wireless technology**

Before JSR 82 came into play, there had been no standardized way to develop Bluetooth applications, while the Bluetooth hardware advanced. JSR82 is the first standard specification to develop Bluetooth applications using the Java language. It provides bluetooth capabilities to J2ME virtual machine devices. A set of Java APIs abstract the complexity of Bluetooth protocol stack into classes, interfaces and methods.

This way we can concentrate our efforts on applications development, rather than the low level details of Bluetooth.

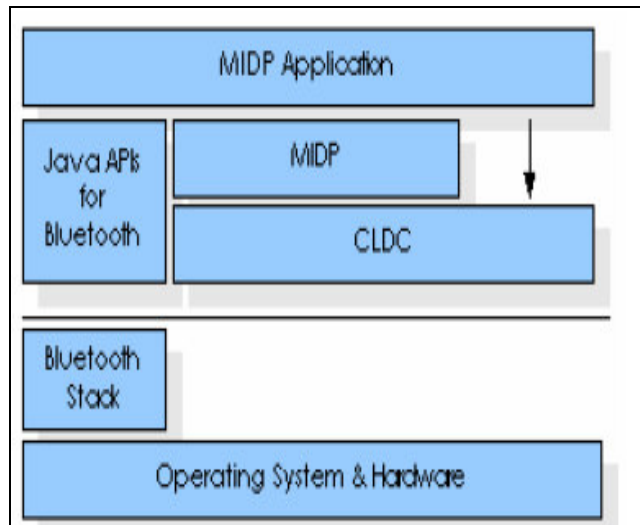


Figure 3.1 Organization of CLDC, MIDP and APIs

JSR 82 API provides us a means to discover and register devices and services. It establishes, manages and controls RFCOMM, L2CAP and OBEX connections among devices, which are used for data communication. Apart from this, JSR 82 provides security for these activities.

### 3.1. Organization and Packages

The Java APIs for bluetooth targeted devices have the following characteristics: 512 KB of memory, bluetooth wireless network connection and implementation of CLDC. Therefore JSR82 consist of two optional packages. They are the Core Bluetooth API, named *javax.bluetooth*, and the Object Exchange package, named *javax.obex*. Both packages extend the functionality of and depend on the Core CLDC Generic Connection Framework, named *javax.microedition.io* and MIDP2.0. The structure of packages provided by the CLDC configuration, the MIDP 2.0 profile and JSR82 is described as follows in the tables:

CLDC Packages	Functionality
Java.microedition.io	Generic connection Framework classes to provide network support
Java.lang	A subset of core Java programming language classes
Java.util	Subset of utility classes
Java.io	System input and ouput classes through data streams

Table 1. Classes and interfaces of Connection Limited Device Configuration

<b>MIDP packages</b>	<b>Functionality</b>
Javax.microedition.lcdiui	User interface classes
Javax.microedition.lcdiui.game	Gaming classes
Javax.microedition.lcdiui.media	Interfaces for controlling audio and sound classes
Javax.microedition.lcdiui.media.control	Sound control classes
Javax.microedition.pki	Public key classes for authentication
Javax.microedition.rms	Persintece classes for storage and to retrieve information
Javax.microedition.midlet	Application midlet interface

Table 2. Classes and interfaces of Mobile Interface Device Profile

<b>JSR 82 Packages</b>	<b>Functionality</b>
Javax.Bluetooth	The core package for building wireless applications upon bluetooth technology
Javax.Obex	Optional package for object exchange

Table 3. Classes and interfaces provided by JSR 82

### **3.2 Bluetooth System Requirements**

The Java APIs require from the supporting Bluetooth protocols and profiles to satisfy a set of requirements:

1. It must be qualified according to the Bluetooth qualification Programm, for at least the Generic Access Profile, Service Discovery Application Profile and Serial Port Profile.
2. The system must support three communication layers as defined in the 1.1 Bluetooth Specification, and the implementation of this API must have access to them : Service Discovery Protocol (SDP), Radio Frequency Communication protocol (RFCOMM) and Logical Link Adaptation Protocol (L2CAP) layers.
3. The system must also provide a Bluetooth Control Center. As many applications may execute concurrently, BCC provides application from harming each other.
4. BCC capabilities allow to configure some parameters in the Bluetooth stack. Though it is not defined in tis specification, it is an important part of its security architechture

JSR82 was defined using J2ME APIs and CLDC/MIDP. The specification has the next characteristics. It gives support and APIs for protocols of bluetooth and profiles. JSR82 include in its APIs support for OBEX, L2CAP and RFCOMM communication protocols. It addresses the Generic Access Profile, the Service Discovery Application Profile, the Serial Port Profile and Generic Object Exchange Profile as well as the Service Discovery Protocol. It is then required from the underlying stack a proper implementation giving access to these protocols, layers and profiles

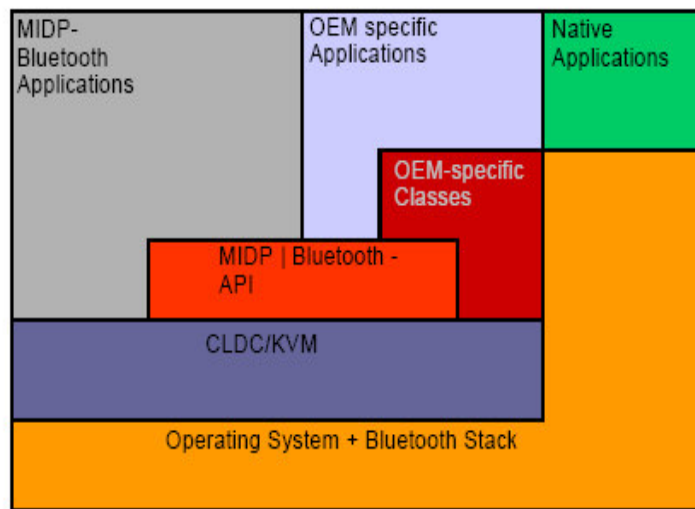


Figure 3.2. Bluetooth API is extensible.

Other Bluetooth profiles are to be built upon this Java specification if the core specification of th JSR82 does not change, using Java programming. That makes JSR 82 flexible and extensible. That is, developers can build their own API on top of this API

### **3.3 Application programming**

An end user application conforming to the MIDP specification is called a MIDlet. The programming interfaces available to a JSR82 enabled MIDlet are sketched in the figure.

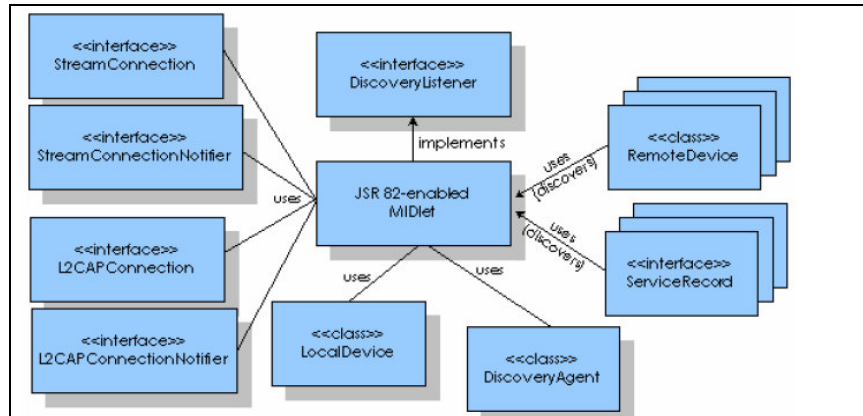


Figure 3.3 MIDlet’s upon Bluetooth API anatomy

Programmers of end users applications will make use of this interfaces for the next purposes: initializing the protocol stack, discovering devices and services, opening connections, closing connections, sending and receiving data and different application performing as sketched in the figure:

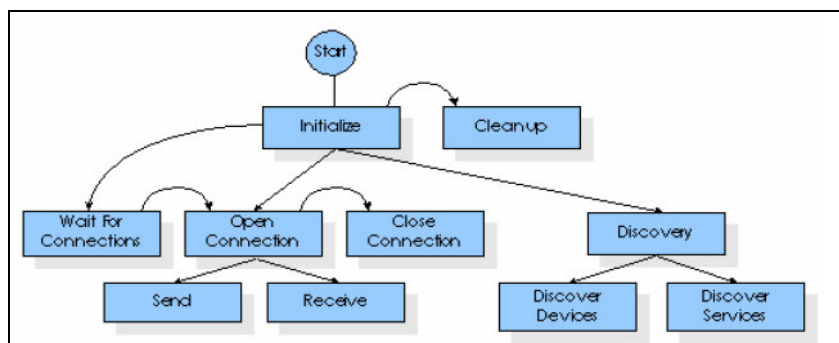


Figure 3.4 Functionality of the API

### 3.3.1 Bluetooth Stack initialization

Before anything else is done, the bluetooth protocol stack, which is responsible for controlling the device, has to be started with the aim to make the device ready for wireless communication. The implementation of the BCC is left to vendors, and they initialize the protocol stack in different ways. These BCC changes may not be likely to be changed. There might be an API providing an interface for the initialization but not part of the JSR 82, like as follows:

```

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

### **3.3.2 Device Management**

The functions to provide device management capabilities are provided by the next java classes: *java.bluetooth.LocalDevice*, *javax.bluetooth.remotedevice* together with the *javax.bluetooth.DeviceClass*.

#### **3.3.2.1 Local Device**

Local devices are represented by objects of the class *javax.bluetooth.Localdevice*. This class allows us to get and manage information about the local device. The main methods in this class are :

*LocalDevice getLocal Device()* fetches the object that represents the local device

*String getAddress()* gets the Bluetooth address of the local device

*Boolean setDiscoverabel()* places the device in the discovery mode

*DiscoveryAgent getDiscoveryAgent* fetches the discovery agent object

#### **3.3.2.2 Remote Device**

In a similar way, the class *RemoteDevice* represents the remote device, providing a means to contain information about the remote device such as the name, the address and other terms associated with a connection, as well as methods regarding security. Some methods in this class are:

*RemoteDevice getRemoteDevice (javax.microedition.io.Connection)* fetches the remote object associated with the correspondent connection

*String getBluetoothAddress()* gets the address of the remote device

*String getFriendlyname()* fetches the name of the remote device

*Boolean authenticate()* attempts to authenticate the remote device

*Boolean isAuthenticated()* informs whether the remote device is authenticated

*Boolean isEncrypted()* tells if the remote devices uses encryption

#### **3.3.3 Service Registration**

In order for the services to be discovered, servers of applications must register whichever the services they want to offer. This way they are published or advertised. For that, several steps are to be accomplished by the server: creating a describing record of the service and adding it to the Service Discovery Database (SDDB), managing this database by updating, registering the security methods regarding this service, accepting requests of connections

A *Connector.open()* call creates a record of the service automatically. A call to the *StreamconnectionNotifier* or *L2CAPconnectionNotifier* register the service in the SDDB.

### **3.3.4 Connection**

Between the communication entities, there must be a common communication protocol. JSR 82 allows connection to every application service held by RFCOMM, OBEX or L2CAP

#### **3.3.4.1 Serial Port Profile**

As a stream based interface to this RFCOMM protocol, Java provides the SPP, that is the Serial Port Profile, facilitating communication and emulating a serial connection. Some capabilities and limitations are: only one RFCOMM session is held at a time between two devices. Up to 60 logical serial connections may be multiplexed over this session. At most, 30 RFCOMM services may a device have. A device can have only one client connection to a particular service.

If two devices want to use the SPP in order to set up and hold a communication, some steps, either for the server or for the client are to be followed.

The server is supposed to:

1. Build up a Uniform Resource locator and place it in the service record
2. Accepting connections from the client
3. Sending and Receiving data

The client entity:

1. Initiate a service discovery to retrieve the record of services
2. Get the URL
3. Open a connection
4. Sendig and receiveing data

### **3.3.5 Device and Service Discovery**

MIDlets need a mechanism to find another devices and services around. This service is provided by the classes *discoveryAgent*, *discoveryListener*, *RemoteDevice* and the *ServiceRecord* inside the core Bluetooth API, allowing the MIDlet to perform device and service discovery.

At first, MIDlets call the method *DiscoveryAgent.startInquiry* to place the device in inquiry mode. The device calls back the MIDlet whenever a device or a service is discovered by calling back the methods *deviceDiscovered()* or *serviceDiscovered()*. The methods *retrieveDevices()* and *searchServices()* search in the local cache for previously retrieved services. The figure sketches the stated above:

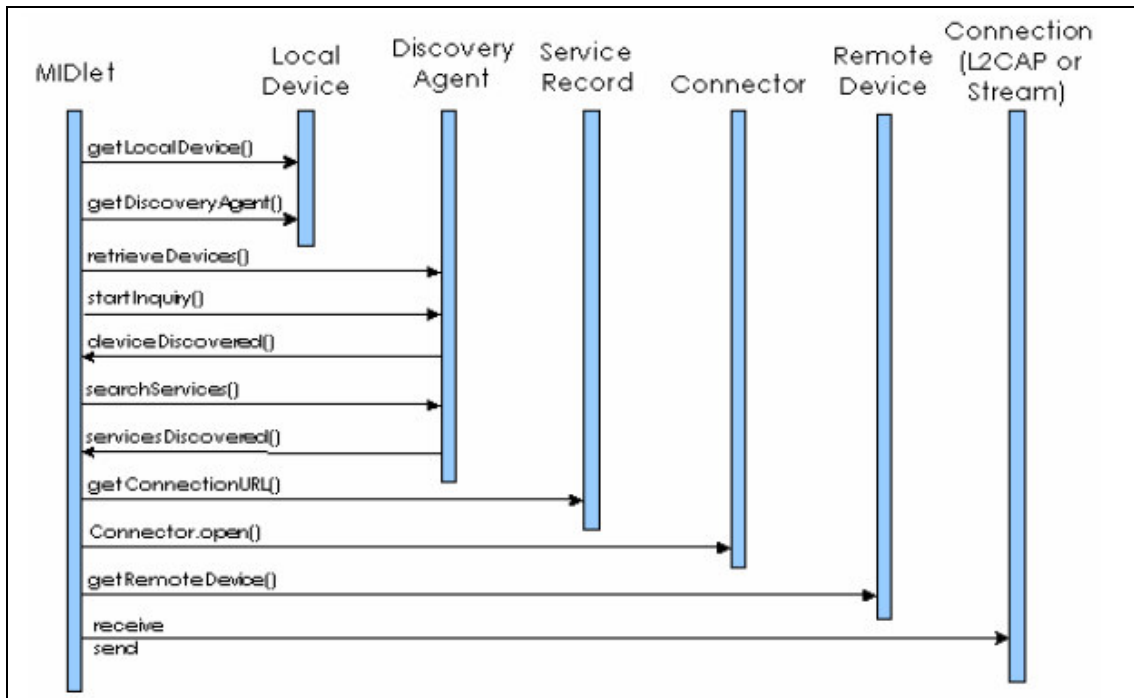


Figure 3.5. Running the MIDlet

#### **4. CONCLUSIONS**

On the one hand, Bluetooth has a potentially huge market due to the huge expansion of mobile phones, among other reasons. This makes Bluetooth an attractive market for vendors of software. Bluetooth works together with Java to build and run wireless applications upon mobile phones. Java developers have been able to develop a Java independent platform (J2ME) for small devices. This Platform is independent from the other bigger Java platforms and therefore, it is now possible to exploit fully the power of the Java programming language without requiring a computer or powerful processors, big memories, persistent storage...

On the other hand, an Application Programmer Interface (JSR 82) is provided for user application programmers make programming upon bluetooth enabled electronic devices easier. This API is a key that helps software developers to construct new programmes and APIs in a standardized way, tapping the attractive bluetooth market. All this is made in a suitable structured way, in a organized packaged structured providing extensibility and flexibility.

## **5. REFERENCES**

As follows a list of references where information was taken for the paper. If you have more interest in the task of the seminar, I would recommend to check the references out to find further information.

<http://java.sun.com/j2me>

<http://www.jcp.org/en/jsr/overview>

<http://www-106.ibm.com/developerworks/java/library/j-j2me/>

<http://developers.sun.com/techttopics/mobility/midp/articles/bluetooth2/>

<http://developers.sun.com/techttopics/mobility/midp/articles/bluetooth1/>

<http://www.palowireless.com/infotooth/tutorial/profiles.asp>

[http://www.widcomm.com/Products/DS\\_bte.pdf](http://www.widcomm.com/Products/DS_bte.pdf)

[http://www.widcomm.com/Products/bluetooth\\_comm\\_software\\_bte.asp](http://www.widcomm.com/Products/bluetooth_comm_software_bte.asp)

[http://www.m-indya.com/mwap/bluetooth/bluetooth\\_components.htm](http://www.m-indya.com/mwap/bluetooth/bluetooth_components.htm)

<http://www.microjava.com/articles/Bluetooth-jsr-82-training.pdf>

<http://zone.ni.com/devzone/conceptd.nsf/webmain/>

<http://developers.sun.com/techttopics/mobility/getstart/articles/survey/>

<http://developers.sun.com/techttopics/mobility/midp/articles/genericframework/>

