STUDY OF BLUETOOTH WIRELESS TECHNOLOGY USING JAVA

Mrs. Pratibha Singh

Dept. of Computer Science & Engg. , RIT, Raipur

Chhattisgarh, India

er.pratibha@yahoo.in

Mr.Dipesh Sharma

Dept. of inf.technology, RIT, Raipur,

Chhattisgarh, India

download.dks@gmail.com

Mr. Sonu Agrawal

Dept. of Computer Science & Engg. , SSCET, DURG,

Raipur, (Chhattisgarh), India

agrawalsonu@gmail.com

ABSTRACT

Bluetooth is a wireless communication protocol. Since it's a communication protocol, we can use Bluetooth to communicate to other Bluetooth-enabled devices. In this sense, Bluetooth is like any other communication protocol that you use every day, such as HTTP, FTP, SMTP, or IMAP. Bluetooth has a client-server architecture; the one that initiates the connection is the client, and the one who receives the connection is the server. Bluetooth is a great protocol for wireless communication because it's capable of transmitting data at nearly 1MB/s, while consuming 1/100th of the power of Wi-Fi.Bluetooth is a short-range universal wireless connectivity standard for electronic appliances and mobile devices.Imagine being able to use your Bluetooth-enabled mobile phone to lock and unlock your car, operate your garage door, and control your TV, VCR, DVD player, and other consumer appliances. If you want to make that kind of control available to your users, you'll need to be able to write Bluetooth applications . The purpose of this article is to give a good introduction that how to use java for the Bluetooth wireless technology , including an overview of its profiles. i'll also cover the the classes and methods of JSR-82, the official Java Bluetooth API. Finally, i'll wrap things up by describing what software that i'll need.

GENERAL TERMS
Bluetooth, Java, Java API, MANET

I. INTRODUCTION

Wireless is a term used to describe telecommunications in which electromagnetic waves (rather than some form of wire) carry the signal over part or all of the communication path and the network is the totality of switches, transmission links and terminals used for the generation, handling and receiving of telecoms traffic. Wireless networks are rapidly evolving, and are playing an increasing role in the lives of people throughout the world and ever-larger numbers of people are relying on the technology directly or indirectly.

The area of wireless communications is an extremely rich field for research, due to the difficulties posed by the wireless medium and the increasing demand for better and cheaper services. As the wireless market evolves, it is likely to increase in size and possibly integrate with other wireless technologies, in order to offer support for mobile computing applications, of perceived performance equal to those of wired communication networks. Wireless Networks aims to provide an excellent introductory text covering the wireless technological alternatives offered today. It will include old analog cellular systems, current second generation (2G) systems architectures supporting voice and data transfer and also the upcoming world of third generation mobile

networks. Moreover, the modern wireless technology topics, such as Wireless Local Loops (WLL), Wireless LANs, Wireless ATM and Personal Area Networks (such as Bluetooth).

A Bluetooth connection is the result of a complex device pairing process, and provides a channel on which many data services can be provided, such as voice, internet communication, file sharing, printer connection etc. One of the downsides are that these connections are point to point, meaning that a device beyond the relatively short Bluetooth distance of
approximately 10 meters will not be able to detect a given service.The Bluetooth using java framework provides broadcasting over several hops, in which one scenario (depicted in Figure 1) enables a phone 50 meters away from a printer to use it, by routing through other mobile phones and laptops (all running BEDnet).The framework, developed in J2ME using the JSR-82 API, acts as a middleware between the Bluetooth API and the application. The framework will automatically detect new devices, devices that have left the network and use a MANET routing protocol to efficiently route packets.In order for Bluetooth devices to communicate properly, they all need to conform to the Bluetooth specification. The Bluetooth specification, like any other spec, defines the standard that a Bluetooth device should adhere to, as well as rules that need to be enforced when communicating. The Bluetooth protocol stack and profiles together comprise the Bluetooth specification.

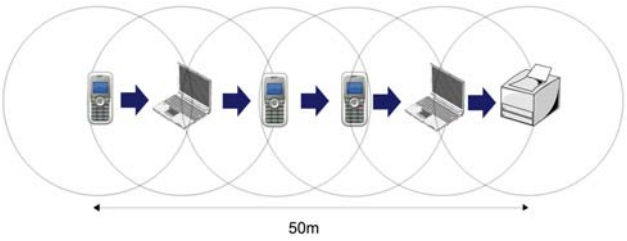

fig 1: A real multihop routing scenario**.**

II. RELATED WORKS

Most of existing work done in this field focuses on parts of subject eg. Framework [1], different scatternet protocols[2],programming solution ,[3],[4],bluetooth technology and optimization[5],and p2p routing.

III. ENABLING TECHNOLOGIES

Here I describe key aspects of four basic technologies: Bluetooth, Java for Mobile Devices, Java Bluetooth Application(API), and Mobile Ad-Hoc Network Routing Protocols.

A. BLUETOOTH

Bluetooth is an emerging low cost and low power short-range radio technology. It has been projected that as many as 200 million Bluetooth devices will be shipped in year 2001. Thus, Bluetooth is likely to become another important platform for ad hoc networking.Ad hoc networking over Bluetooth can lead to many useful applications. ***For example***, in a conference room, a special announcement can be broadcast to the Blue-tooth enabled mobile phones and hand-held computers through an ad hoc network. Bluetooth ad hoc networks can also be used for rapid deploy-ment of EMID (electromagnetic identification) readers .It is basically organized as small group of device called piconets.each piconet consists of one master device and upto seven active slave devices. Connecting two devices via bluetooth requires two phases:

1) INQUIRY: In this process sender broadcasting inquiry packets, which do not contain the identity of sender. And in Inquiry Scan receiver devices listen for packets send by sender, and upon detection of any such packets, the device broadcasts an inquiry response packet.

2) PAGE: When paging, a sender device tries to form a connection with a device whose identity and clock are known. Page packets are sent, which contain the sender's device address and clock, for synchronization. And in Page Scan that is perform after paging: In this state a receiver device listens for page packets. Receipt is acknowledged and Synchronization between the devices is established.

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

- The underlying system must be "qualified," in accordance with the Bluetooth Qualification Program, for at least the Generic Access Profile, Service Discovery Application Profile, and Serial Port Profile.
- The system must support three communication layers or protocols 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 Communications Protocol (RFCOMM), and Logical Link Control and Adaptation Protocol (L2CAP).
- The system must provide a Bluetooth Control Center (BCC), a control panel much like the application that allows a user or OEM to define specific values for certain configuration parameters in a stack.

OBEX support can be provided in the underlying Bluetooth system or by the implementation of the API. The OBEX protocol provides support for object exchanges, and forms the basis for Bluetooth profiles such as the Synchronization Profile and the File Transfer Profile.

B. JAVA FOR MOBILE DEVICES

Java ME technology was originally created in order to deal with the constraints associated with building applications for small devices. the basics for Java ME technology to fit such a limited environment and make it possible to create Java applications running on small devices with limited memory, display and power capacity. Java ME platform is a collection of technologies and specifications that can be combined to construct a complete Java runtime environment specifically to fit the requirements of a particular device or market. This offers a flexibility and co-existence for all the players in the eco-system to seamlessly cooperate to offer the most appealing experience for the end-user.

The Java ME technology is based on three elements:

i. A configuration provides the most basic set of libraries and virtual machine capabilities for a broad range of devices,

ii. A profile is a set of APIs that support a narrower range of devices, and

iii. An optional package is a set of technology-specific APIs.

Over time the Java ME platform has been divided into two base configurations, one to fit small mobile devices and one to be targeted towards more capable mobile devices like smart-phones and set top boxes.

The configuration for small devices is called the Connected Limited Device Configuration (CLDC) and the more capable configuration is called the Connected Device Configuration (CDC).

The figure 2 below represents an overview of the components of Java ME technology and how it relates to the other Java Technologies.

C. JAVA BLUETOOTH API

Bluetooth hardware has advanced, there has been no standardized way to develop Bluetooth applications - until JSR 82 came into play. It is the first open, non-proprietary standard for developing Bluetooth applications using the Java programming language. It hides the complexity of the Bluetooth protocol stack behind a set of Java APIs that allow you 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. Like all JSRs, the Java APIs for Bluetooth are being developed through the Java Community Process. JSR 82 consists of 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.

Note: The Java APIs for Bluetooth do not implement the Bluetooth specification, but rather provide a set of APIs to access and control a Bluetooth-enabled device. JSR 82 concerns itself primarily with providing Bluetooth capabilities to J2ME-enabled devices.
The Java APIs for Bluetooth target devices with the following characteristics:

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

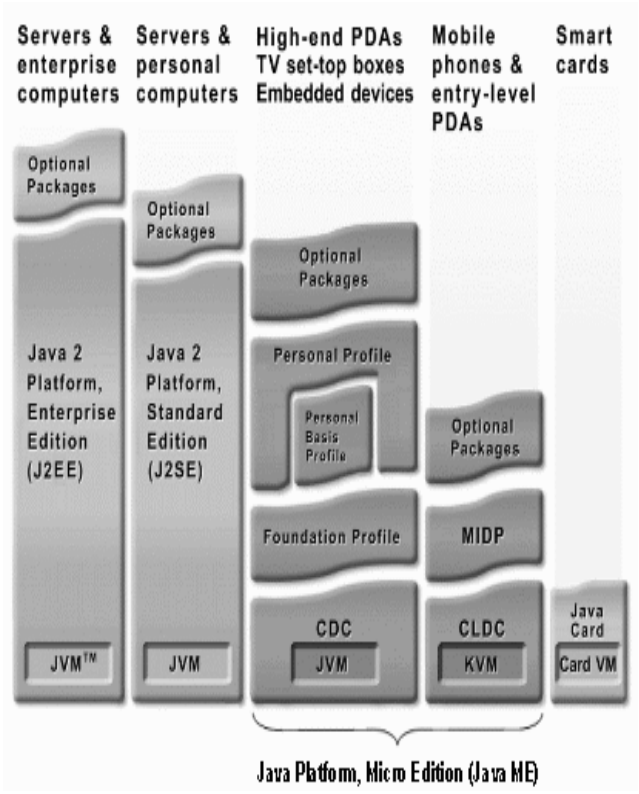

Fig 2: java micro edition(java ME)

THE API ARCHITECTURE

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

- 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.
- 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 JSR 82 specification addresses the Generic Access Profile, Service Discovery Application Profile, Serial Port Profile, and Generic Object Exchange Profile.
- 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.

JSR-82 requires that the Bluetooth stack underlying a JSR-82 implementation be qualified for the Generic Access Profile, the Service Discovery Application Profile, and the Serial Port Profile. The stack must also provide access to its Service Discovery Protocol, and to the RFCOMM and L2CAP layers. 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 restricted 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. Figure 3 shows where the APIs defined in this specification fit in a CLDC/MIDP architecture.

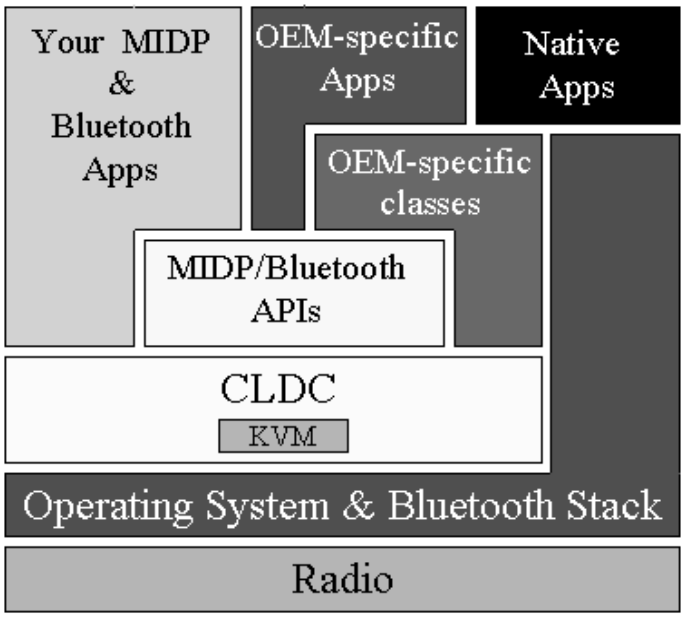

Fig3. High-level Architecture of J2ME CDLC/MIDP and Bluetooth

D.MOBILE AD HOC NETWORKS (MANET'S)

The main characteristics of MANET are:

- Mobile wireless network, it is capable of autonomous operation.
- It operates without base station infrastructure.
- Its nodes cooperate to provide connectivity.
- It operates without centralized administration.
- Its nodes cooperate to provide services.

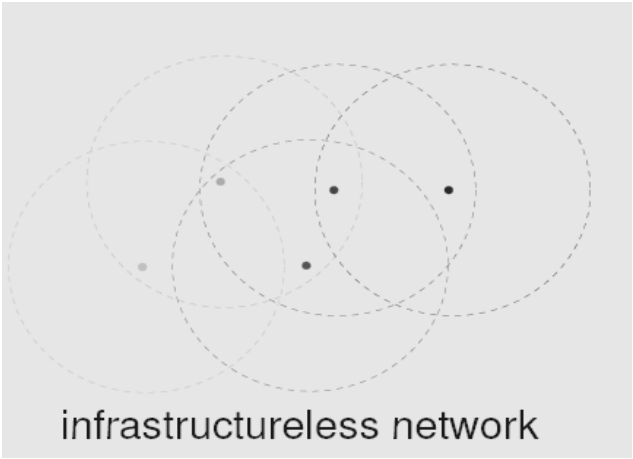

Fig.**4** ad hoc routing

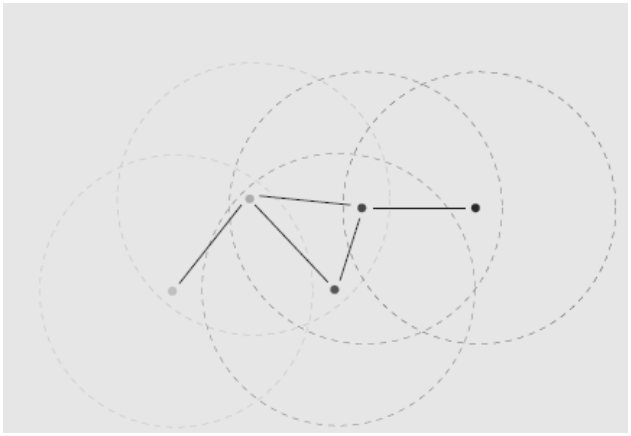

Fig 5.ad hoc routing(equivalent topology)

IV. IMPLEMENTATION CONCEPTS

Before getting started with java over bluetooth I explain something about bluetooth stack and bluetooth profile.

1. THE BLUETOOTH STACK

A Bluetooth stack is required in order for a Bluetooth host (the PC) to properly communicate to the Bluetooth device (the controller). the bottom layer of the stack is the Host Controller Interface! The Host Controller Interface is literally the software required to interface the Bluetooth host and the Bluetooth device (the controller).
The Bluetooth stack is the software or firmware component that has direct access to the Bluetooth device. It has control over things such as device settings, communication parameters, and power levels for the Bluetooth device. The stack itself consists of layers, and each layer of the stack has a specific task in the overall functionality of the Bluetooth device. Since Bluetooth device manufacturers are not required to use all of the layers in the stack, we're only going to cover the main ones that are implemented in almost every Bluetooth device.

- HCI is the Host Controller Interface. This layer is the interface between the radio and the host computer.
- L2CAP stands for Logical Link Controller Adaptation Protocol. This layer is the multiplexer of all data passing through the unit. Audio signals, however, have direct access to the HCI.
- SDP is the Service Discovery Protocol. The SDP layer is used to find services on remote Bluetooth devices.
- RFCOMM is widely known as the virtual serial port protocol.
- OBEX is the object exchange protocol.

2. BLUETOOTH PROFILE

To use Bluetooth wireless technology, a device must be able to interpret certain Bluetooth profiles, which are definitions of possible applications and specify general behaviors that Bluetooth enabled devices use to communicate with other Bluetooth devices. There are a wide range of Bluetooth profiles that describe many different types of applications or use cases for devices.A Bluetooth profile is a designed set of functionality for Bluetooth devices. For instance, using the examples listed , the phone and the PDA must both support the Synchronization Profile in order to synchronize data between themselves. To send object data like a .vcf from the PDA to the phone, both devices need to have the Object Push Profile implemented. Finally, the PDA and the wireless phone must both support the Dialup Networking Profile in order for the PDA to wirelessly browse the Internet from the phone. If you want your Bluetooth-enabled devices to interact, having a Bluetooth stack is not good enough: they need to conform to a particular profile. A word of caution here: do not get Bluetooth profiles confused with J2ME profiles. J2ME profiles are a set of Java classes that extend the functionality of a J2ME Configuration. For instance, the MID Profile is a set of Java classes that extend the functionality of the Connected Limited Device Configuration. On the other hand, a Bluetooth profile can be implemented in any language and on any platform, because it refers to a defined set of functionality for a Bluetooth-enabled device. So the Object Push Profile can be implemented on a Palm OS PDA in C++, and can be implemented on a

Bluetooth-enabled printer in assembly language. For those of you who are familiar with RUP methodology, Bluetooth Profiles are also called Bluetooth Use Cases.

3. PACKAGES

The Java APIs for Bluetooth define 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.

Again, the OBEX APIs are defined independently of the Bluetooth transport layer and packaged separately. Each of the above packages represents 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 incorporate this specification.

Java Bluetooth Application Programming Concepts

The anatomy of a Bluetooth application has five parts:

- Stack Initialization,
- Device Management,
- Device Discovery,
- Service Discovery,
- Communication.

The Java Bluetooth Specification adds a special component to the mix called the Bluetooth Control Center (BCC), NOW there is arising a question about BCC so I explain it that Bluetooth devices that implement this API may allow multiple applications to execute concurrently. The BCC prevents any application from harming another. The BCC is a set of capabilities that allow a user or OEM to resolve conflicting application requests by defining specific values for certain configuration parameters in a Bluetooth stack. It is the central authority for local Bluetooth device settings. The BCC might be a native application, an application with a separate API, or simply a group of settings that are specified by the manufacturer and cannot be changed by the user. Note that the BCC is not a class or an interface defined in this specification but an important part of its security architecture.

Stack Initialization

The Bluetooth stack is responsible for controlling the Bluetooth device, so you need to initialize the Bluetooth stack before you can do anything else. The initialization process comprises 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 an application with a GUI interface, and on another it may be a series of settings that cannot be changed by the user.

```
import javax.bluetooth.*;

import javax.microedition.io.*;
import com.atinav.BCC;

public class WirelessDevice implements DiscoveryListener {
   LocalDevice localDevice = null;

   public WirelessDevice (){
      //setting the port number using Atinav's BCC
      BCC.setPortName("COM1");

      //setting the baud rate using Atinav's BCC
      BCC.setBaudRate(57600);

      //connectable mode using Atinav's BCC
      BCC.setConnectable(true);
```

```
        //Set discoverable mode using Atinav's BCC
        BCC.setDiscoverable(DiscoveryAgent.GIAC);

        try{
            localDevice = LocalDevice.getLoaclDevice();
        }
        catch (BluetoothStateException exp) {
        }

        // implementation of methods in DiscoveryListener class
        // of javax.bluetooth goes here

        // now do some work
    }
}
```

Device Management

LocalDevice and Remote Device are the two main classes in the Java Bluetooth Specification that allow you to perform Device Management. These classes give you the ability to query statistical information about your own Bluetooth device (LocalDevice) and information on the devices in the area (RemoteDevice).The static method LocalDevice.getLocalDevice() returns an instantiated LocalDevice object for you to use. In order to get the unique address of your Bluetooth radio, just call get BluetoothAddress() on your local device object. The Bluetooth address serves the same purpose of the MAC address on the network card of your computer; every Bluetooth device has a unique address. If you want other Bluetooth devices in the area to find you, then call the set Discoverable() method in LocalDevice object. The following code snippet retrieves that information for the local device.

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

You can get the same information about a remote device:

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

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

Device Discovery

Bluetooth device has no idea of what other Bluetooth devices are in the area. Perhaps there are laptops, desktops, printers, mobile phones, or PDAs in the area. Who knows? The possibilities are endless. In order to find out, Bluetooth device will use the Device Discovery classes that are provided into the Java Bluetooth API in order to see what's out there. Let's take a look at the two classes needed in order for your Bluetooth device to discover remote Bluetooth devices in the area: DiscoveryAgent and DiscoveryListener. After getting a Local Device object, just instantiate a DiscoveryAgent by calling LocalDevice.getDiscoveryAgent().

```
LocalDevice localdevice = LocalDevice.getLocalDevice();
DiscoveryAgent discoveryagent = localdevice.getDiscoveryAgent();
```

Their are multiple ways to discover remote Bluetooth devices, but to be brief, I'll just show you one particular way. First, object must implement the DiscoveryListener interface. This interface works like any listener, so it'll notify when an event happens. In this case, i'll be notified when Bluetooth devices are in the area. In order to start the discovery process, just call the startInquiry() method on DiscoveryAgent. This method is non-blocking, so you are free to do other things while you wait for other Bluetooth devices to be found.

When a Bluetooth device is found, the JVM will call the deviceDiscovered() method of the class that implemented the DiscoveryListenerinterface. This method will pass a RemoteDevice object that represents the device discovered by the inquiry.

Service Discovery

Now how to find other Bluetooth devices, it would be really nice to see what services that those devices offer. Of course, if the RemoteDevice is a printer, then it can offer a printing service. But what if the RemoteDevice is a computer? Would it readily come to mind that you can also print to a printer server? That's where Service Discovery comes in. I can never be sure what services a RemoteDevice may offer; Service Discovery allows to find out what they are.

Service Discovery is just like Device Discovery in the sense that use the DiscoveryAgent to do the "discovering." The searchServices()method of the DiscoveryAgent class allows to search for services on a RemoteDevice. When services are found, theservicesDiscovered() will be called by the JVM if object implemented the DiscoveryListener interface. This callback method also passes in a ServiceRecord object that pertains to the service for which searched. With a ServiceRecord in hand, i can do plenty of things, but would most likely would want to connect to the RemoteDevice where this ServiceRecord originated:

```
String connectionURL = servRecord[i].getConnectionURL(0, false);
```

Service Registration

Before a Bluetooth client device can use the Service Discovery on a Bluetooth server device, the Bluetooth server needs to register its services internally in the Service Discovery database (SDDB). That process is called Service Registration. This section will discuss what's involved for Service Registration for a Bluetooth device, and I'll also give you a rundown of the classes needed to accomplish this.

Note: In a peer-to-peer application, such as a file transfer or chat application, be sure to remember that any device can act as the client or the server, so you'll need to incorporate that functionality (both client and server) into your code in order to handle both scenarios of Service Discovery (i.e., the client) and Service Registration (i.e., the server). Here's a scenario of what's involved to get your service registered and stored in the SDDB.

1. Call Connector.open() and cast the resulting Connection to a StreamConnectionNotifier.
2. Use the LocalDevice object and the StreamConnectionNotifier to obtain the ServiceRecord that was created by the system.
3. Add or modify the attributes in the ServiceRecord (optional).
4. Use the StreamConnectionNotifier and call acceptAndOpen() and wait for Bluetooth clients to discover this service and connect.
   i) The system creates a service record in the SDDB.
5. Wait until a client connects.

6. When the server is ready to exit, call close() on the StreamConnectionNotifier.
   ii) The system removes the service record from the SDDB.

StreamConnectionNotifier and Connector both come from the javax.microedition.io package of the J2ME platform. The code that accomplishes the above task is shown below in the following snippet:

```
// lets name our variables
StreamConnectionNotifier notifier = null;
StreamConnection sconn = null;
LocalDevice localdevice = null;
ServiceRecord servicerecord = null;

// step #1
// the String url will already be defined with the
// correct url parameters
notifier = (StreamConnectionNotifier)Connector.open(url);

// step #2
// we will get the LocalDevice if not already done so
localdevice = LocalDevice.getLocalDevice();
servicerecord = localdevice.getRecord(notifier);

// step #3 is optional

// step #4
// this step will block the current thread until
// a client responds this step will also cause the
// service record to be stored in the SDDB
notifier.acceptAndOpen();

// step #5
// just wait...
// assume the client has connected and you are ready to exit

// step #6
// this causes the service record to be removed
// from the SDDB
notifier.close();
```

And that's all that you need to do Service Registration in Bluetooth. The next step is Communication.

Communication

For a local device to use a service on a remote device, the two devices must share a common communications protocol. So that applications can access a wide variety of Bluetooth services, the Java APIs for Bluetooth provide mechanisms that allow connections to any service that uses RFCOMM, L2CAP, or OBEX as its protocol. If a service uses another protocol (such as TCP/IP) layered above one of these protocols, the application can access the service, but only if it implements the additional protocol in the application, using the CLDC Generic Connection Framework. Because the OBEX protocol can be used over several different transmission media - wired, infrared, Bluetooth radio, and others - JSR 82 implements the OBEX API (javax.obex) independently of the core Bluetooth API (javax.bluetooth). The OBEX API is a separate optional package i can use either with the core Bluetooth package or independently.Bluetooth is a communication protocol, so how do i communicate with it? Well, the Java Bluetooth API gives three ways to send and receive data, but for right now, i'll cover only one of them: RFCOMM.

Note: RFCOMM is the protocol layer that the Serial Port Profile uses in order to communicate, but these two items are almost always used synonymously.

Server Connections with the Serial Port Profile

The RFCOMM protocol, which is layered over the L2CAP protocol, emulates an RS-232 serial connection. The Serial Port Profile (SPP) eases communication between Bluetooth devices by providing a stream-based interface to the RFCOMM protocol. Some capabilities and limitations to note:

- Two devices can share only one RFCOMM session at a time.
- Up to 60 logical serial connections can be multiplexed over this session.
- 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.

The code listing below demonstrates what is needed to open a connection on a Bluetooth device that will act as a server.

```
// let's name our variables

StreamConnectionNotifier notifier = null;
StreamConnection con = null;
LocalDevice localdevice = null;
ServiceRecord servicerecord = null;
InputStream input;
OutputStream output;

// let's create a URL that contains a UUID that
// has a very low chance of conflicting with anything
String url =
  "btspp://localhost:00112233445566778899AABBCCDDEEFF;name=serialconn";
// let's open the connection with the url and
// cast it into a StreamConnectionNotifier
notifier = (StreamConnectionNotifier)Connector.open(url);

// block the current thread until a client responds
con = notifier.acceptAndOpen();

// the client has responded, so open some streams
input = con.openInputStream();
output = con.openOutputStream();

// now that the streams are open, send and
// receive some data
```

For the most part, this looks like just about the same code used in Service Registration, and in fact, it is! Service Registration and Server Communication are both accomplished using the same lines of code. Here's a few items that I want to point out. The String url begins with btspp://localhost:, which is required if you're going to use the Bluetooth Serial Port Profile. Next comes the UUID part of the URL, which is00112233445566778899AABBCCDDEEFF. This is simply a custom UUID that I made up for this service; I could have chosen any string that was either 32 bits or 128 bits long. Finally, we have ; name=serialconn in the url String. I could have left off this part, but I want my custom service to have a name, so the actual service record in the SDDB has the following entry:

```
ServiceName = serialconn
```

The implementation has also assigned a channel identifier to this service. The client must provide the channel number along with other parameters in order to connect to a server.

Client Connections with the Serial Port Profile

Establishing a connection with the Serial Port Profile for a J2ME client is simple because the paradigm hasn't changed for J2ME I/O. You simply call Connector.open().

```
StreamConnection con =(StreamConnection)Connector.open(url);
```

i obtain the url String that is needed to connect to the device from the ServiceRecord object that i get from Service Discovery. Here's a more complete listing of code that will show you how a Serial Port Profile client makes a connection to a Serial Port Profile server.

```
String connectionURL = serviceRecord.getConnectionURL(0, false);
StreamConnection con =(StreamConnection)Connector.open(connectionURL);
```

What does a SPP client connection URL look like? If the address of the server is 0001234567AB, the String that the SPP client would look something like this:

```
btspp://0001234567AB:3
```

The 3 at the end of the url String is the channel number that the server assigned to this service when this service was added to the SDDB.

Java Bluetooth Development Kits

The most widely available development kit for Java Bluetooth applications is the J2ME Wireless Toolkit 2.2 . It incorporates a Bluetooth network simulator, and has support for OBEX. If you're targeting JSR-82-enabled Nokia phones, such as the 6600, then you may also want to try out the Nokia Developer's Suite 2.1. Much like Sun's Wireless Toolkit, the Nokia Developer's Suite is free and it also includes a Bluetooth network simulator. The Nokia Developer's Suite supports Windows and Linux platforms. SonyEricsson also makes a free development kit for its P900 Java Bluetooth-enabled phone, which can be found at their developer site. Atinav makes one of the most comprehensive JSR-82 implementations and developer kits with support for J2ME CLDC, J2ME CDC, and J2SE devices. They support numerous RS-232, UART, USB, CF, and PCMCIA Bluetooth devices. Their solution is based on an all-Java stack, and their SDK includes the following profiles: GAP, SDAP, SPP, OBEX, FTP, Sync, OPP, Fax, and Printing -- whew! They make the only JSR-82 implementation for the PocketPC platform, and also support Windows and Linux. Possio makes a JSR-82 development kit that complements their Java Bluetooth-enabled access point, the PX30. The PX30 is a Linux-based access point, and is powered by an Intel XScale processor. It includes Wi-Fi, Bluetooth, and the CDC Foundation Profile.Rococo is famous for making the first Java Bluetooth Simulator, although they also make a Java Bluetooth developer kit for the Palm OS 4 platform. The simulator is currently priced at $1000, and supports the following profiles: GAP, SDAP, SPP, and GOEP. Avetana is a German company that makes the only JSR-82 implementation for the Mac OS X platform. They also provide JSR-82 implementations for Windows and Linux.

V. CONCLUSION

This article presented a tutorial on the Java for Bluetooth wireless technology. The sample code demonstrated how easy it is to develop wireless applications for Bluetooth-enabled devices. The APIs enable you to exploit fully the power of the Java programming language to develop wireless applications in a standard way. This set of APIs is a key enabler that will help software vendors and developers tap the potentially huge market for Bluetooth wireless technology. What have we learned here? Hopefully, we should have a good understanding of what Bluetooth is and how to use it. Before i start communicating to other Bluetooth devices, i need to discover the devices in your vicinity, and search for their services. After all of the preliminaries are out of the way, i can stream data back and forth to any Bluetooth-enabled device in our area, whether it's running Java or not.

ACKNOWLEDGMENTS

I give special thanks to Mrs.Uzma who is HOD of COMPUTER SCIENCE DEPT. of Raipur institute of technology. Whose feedback helped me to improve my article.

REFERENCES

[1] Bisdikian, “A framework for building Bluetooth scatternets: A system design approach,” Pervasive and Mobile Computing 1, pp. 190–212,2005.

[2] Singh Annapurna and Mishra Shailendra , “Performance Analysis of Reactive Routing Protocols in Mobile Ad hoc Networks”, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.8, August 2010.
[3] G. Jayakumar, G. Gopinath , “Performance Comparison of MANET Protocol Based on Manhattan Grid Model”, Journal of Mobile Communications, vol. 2, no. 1, pp. 18-26, 2008.
[4] Sayid Mohamed Abdule, Suhaidi Hassan, Osman Ghazali, Mohammed M. Kadhum , “Pause Time Optimal Setting for AODV Protocol on RPGM Mobility Model in MANETs”, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 1, No. 6, December 2010
[5] M. Sun, C. Chang, and T. Lai, “A Self-routing Topology for Bluetooth Scatternets,” I-SPAN, 2002.
[6] J. . E. Group and jsr 118-comments@jcp.org, Mobile Information Device Profile for Java 2 Micro Edition, Version 2.1. Sun Microsystems, Inc.and Motorola, Inc., 2006.
[7] S. Shah et al, “Performance Evaluation of Ad Hoc Routing Protocols Using NS2 Simulation”, Conf. of Mobile and Pervasive Computing, 2008.
[8] Rashmi Popli* & Ritu Saluja**,” Impact of Mobility Models on Performance of DSR”, International Journal of Information Technology and Knowledge Management January June 2009, Volume 2, No. 1, pp. 149-153
[9] P.-C. Wei, C.-H. Chen, C.-W. Chen, and J.-K. Lee, “Support and optimization of Java RMI over a Bluetooth environment,” Concurrency Computat.: Pract. Exper., pp. 1–21, 2002.
[10] C.Blundo and E. Cristofaro, “Issues related to development of wireless peer-to-peer games in J2ME,” Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, 2006.
[11] R. M. Whitaker, L. Hodge, and I. Chlamtac, “Bluetooth scatternet formation: A survey.” Elsevier - Ad Hoc Networks 3, pp. 403–450, 2005.
[12] C.Blundo and E. Cristofaro, “A Bluetooth-based JXME infrastructure,”Proceedings of the 9th International Symposium on Distributed Objects, Middleware, and Applications, pp. 667–682, 2007.
[13] S. Ali, “The Feasibility of Mobile Adhoc Routing Over Bluetooth - and a discussion about the realism of simulations,” Masters thesis in Computer Science at University College London, pp. 1–62, 2007.
[14] J. Haartsen, “The Bluetooth Radio System,” IEEE
[15] Personal Communications, vol. 7, no. 1, pp. 28–36,Feb. 2000.
[16] P. Johansson, N. Johansson, U. Korner, J. Elg, and G. Svennarp, “Short Range Radio Based Ad-hoc Networking: Performance and Properties,” Proc. Of IEEE ICC’99, pp. 1414–1420, 1999.
[17] M. Kalia, D. Bansal, and R. Shorey, “MAC Scheduling and SAR Policies for Bluetooth: AMaster,” Proc. Workshop on Mobile Multimedia, pp. 384–388, 1999.
[18] M. Kalia, S. Garg, and R. Shorey, “Efficient Policies for Increasing Capacity in Bluetooth: an Indoor Pico-Cellular Wireless System,” Proc. IEEE Vehicular Technology Conference (VTC 2000), pp. 907–911,2000.
[19] S. Jain, R.C. Shah, G. Borriello, W. Brunette, and S. Roy,“Exploiting Mobility for Energy Efficient Data Collection inSensor Networks,” Proc. IEEE Symp. Modeling and Optimization inMobile, Ad Hoc, and Wireless Networks, 2004.
[20] X. Lin, G. Sharma, R.R. Mazumdar, and N.B. Shroff, “DegenerateDelay-Capacity Tradeoffs in Ad-Hoc Networks with Brownian
[21] Mobility,” IEEE/ACM Trans. Networking, vol. 14, pp. 2777-2784,2006.
[22] H. Dubois-Ferriere, M. Grossglauser, and M. Vetterli, “AgeMatters: Efficient Route Discovery in Mobile Ad Hoc Networks Using Encounter Ages,” Proc. MobiHoc ’03, pp. 257-266, 2003.