

**Bluetooth**  
 Introduction and Detailed JSR 82  
 Explanation

Murali Annavaram & Bhaskar  
 Krishnamachari  
 Ming Hsieh Department of Electrical  
 Engineering  
 USC

Lecture notes based in part on slides created by Vidya Settur,  
 Thanks also to Mikko H. Lipasti for course suggestions.

**A Brief Introduction  
 to Bluetooth**

Bhaskar Krishnamachari  
 Ming Hsieh Department of Electrical  
 Engineering  
 USC

**Basics**

- o Named after Hagar Blatand (Bluetooth) 10<sup>th</sup> century King of Denmark
- o Industry standard for wireless personal area networks
- o Bluetooth SIG/IEEE 802.15.1
- o First developed by Ericsson ~1994, SIG formed in 1998. Versions 1 -1.1 - 1.2 - 2.0 - 2.1

**Bluetooth Communication**

- o Frequency-hopping, Time division duplex system
- o 79 1MHz channels in 2.4GHz ISM band, 1600 times a second
- o Output power - Class 1: 100mW, Class 2: 2.5mW, Class 3: 1mW

Source: <http://www.informit.com/articles/article.aspx?p=21324>

**Network Topology**

- o Piconet: Master-slave configuration
  - ❖ One master, up to seven active slaves, 255 parked
  - ❖ Master determines the FH sequence and clock

Source: <http://www.ieee802.org/11/Tutorial/90538S-WPAN-Bluetooth-Tutorial.pdf>

**Network Stack**

Source: [www.cs.umu.se/kurser/TDBD16/VT07/Bluetooth-Tutorial-2001.pdf](http://www.cs.umu.se/kurser/TDBD16/VT07/Bluetooth-Tutorial-2001.pdf)

## Link Modes, Data Rates

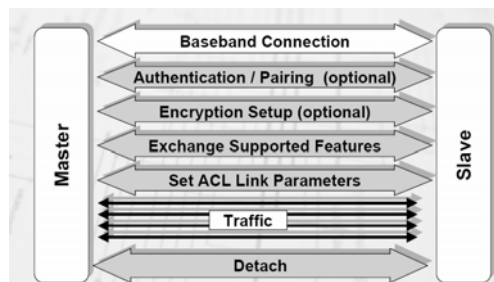
- o Synchronous Connection Oriented and Asynchronous Connection-less modes

Packet type	FEC	Symmetric max rate (kb/s)	Asymmetric max rate (kb/s)	
DM1	2/3	108.8	108.8	108.8
DH1	no	172.8	172.8	172.8
DM3	2/3	258.1	387.2	54.4
DH3	no	390.4	585.6	86.4
DM5	2/3	286.7	477.8	36.3
DH5	no	433.9	723.2	57.6
AUX1	no	185.6	185.6	185.6

Source: www.cs.umu.se/kurser/TDBD16/VT07/Bluetooth-Tutorial-2001.pdf



## ACL Session



Source: www.cs.umu.se/kurser/TDBD16/VT07/Bluetooth-Tutorial-2001.pdf



## Application profiles

- o Standardizes different communication patterns for specific applications; e.g.,
  - ❖ Printing profile
  - ❖ Cordless telephony profile
  - ❖ File transfer profile
  - ❖ Service Discovery profile



## Service Discovery Protocol (SDP)

- o A client server protocol
- o Defines how services are represented
- o How to access service discovery database



## Bluetooth in EE579 Context

- Bluetooth is a wireless communication standard
  - Primarily designed for low power consumption, with a short range
    - **Class 1:** 100 mW ~100 meters
    - **Class 2:** 2.5 mW ~10 meters (**Used in N95**)
    - **Class 3:** 1 mW ~1 meter
- Create a personal-area-network (PAN) with bluetooth devices over a short range
- All devices in the PAN can communicate and share information instantly
- N95 supports V2.0 + EDR ~ 3.0 Mbits/sec
- More info: www.bluetooth.org



## Bluetooth vs Infrared

- N95 also supports Infrared for wireless communication

Infrared	Bluetooth
- Needs line-of-sight	+ No alignment needed
-One-to-one communication	+ Multiple devices can communicate simultaneously
+More secure	- Bluesnarfing & BlueBugging may compromise information
	-Bluetooth is not inherently insecure
	-Need to use strong keys for security



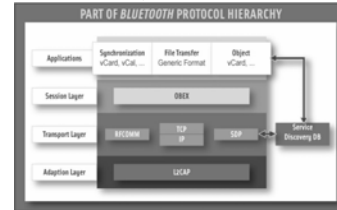
## Bluetooth Naming

- Bluetooth devices use a 48 bit unique address (similar to MAC address)
- Typically use a name, instead of 48 bit address
  - N95 allows users to assign a name to the device
    - Naming is just a simple indirection, different devices can have the same name
  - During discovery process we will use the name to connect



## Bluetooth Transport Protocols

1. RFCOMM: RFCOMM essentially allows transferring bits between two Bluetooth connected device just as if they are connected using a serial port
2. OBEX is a transfer protocol that defines data objects and a communication protocol two devices can use to exchange those objects



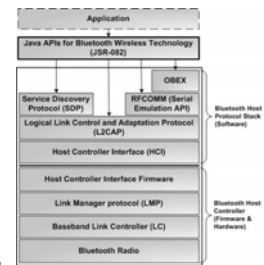
## Bluetooth Connections Steps

1. A Bluetooth service provide device transmit the following sets of information on demand:
  - Device name, class, services
  - Service provider can decide whether or not to make the service discoverable
2. Requestor devices performs a service discovery to find other devices and their services
3. Use of the service from the provider may require pairing
  - Pairing is the process of establishing trust, usually allowing the two parties to share a key
4. Trusted parties can then communicate



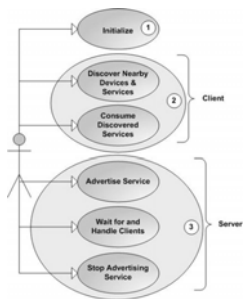
## Bluetooth Support in Java: JSR 82

- JSR 82 exposes the Bluetooth software stack to developers working on the Java platform
  - Service Discovery Protocol (SDP), the Serial Port Profile RFCOMM for serial emulation, Object Exchange APIs.
- We can break down the core Java Bluetooth APIs, found in `javax.bluetooth`, into four categories: initialization, discovery, device management, and communication.



## Bluetooth Usage Model

- **Initialization** – All Bluetooth-enabled applications must first initialize the Bluetooth stack.
- **Client** – A client consumes remote services. It first discovers any nearby devices, then for each discovered device it searches for services of interest.
- **Server** – A server makes services available to clients. It registers them in the Service Discovery Database (SDDb), in effect advertising them. It then waits for incoming connections, accepts them as they come in, and serves the clients that make them. Finally, when the service is no longer needed the application removes it from the SDDb.

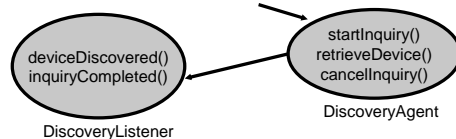


[http://developer.sun.com/mobile/j2se/serial/bluetooth/remote/mugshot\\_en.jpg](http://developer.sun.com/mobile/j2se/serial/bluetooth/remote/mugshot_en.jpg)



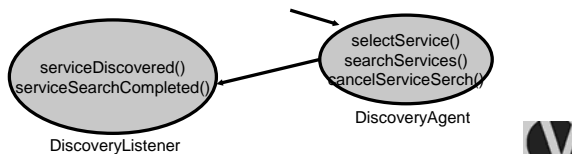
## Discovery Process: Device Discovery

- `DiscoveryAgent` class discovers both devices and services
- `DiscoveryListener` is implemented by clients to get notifications from `DiscoveryAgent`
- Client initiate device discovery using `startInquiry()`
- `DiscoveryAgent` invokes the callback methods `deviceDiscovered()` and `inquiryCompleted()`



## Discovery Process: Service Discovery

- Each device can offer multiple services; hence DiscoverAgent also provides API for service discovery
- selectService() starts serviceDiscovery
- serviceDiscovered(), serviceSearchCompleted() are the callbacks



## Discover Services with UUID

### 1. UUID : Universally Unique Identifier

- Service provider calls uuidgen() to create the UUID
- Client use UUID when searching for service

### 2. Service Discovery Database & Service Discovery Protocol

- Client can use SDP to call SDP in a server which in turn accesses SDB to discover services
- Server creates a service record and inserts into SDDB
- Clients querying the remote SDDB for available services will find the service record

## Device Management

- LocalDevice() class is called by the client application to find information about its own Bluetooth services
  - setDiscoverable()
  - updateRecord()
  - getBluetoothAddress()
  - getProperty()
- RemoteDevice() class is called by the client to get information about the service provider
  - getBluetoothAddress()
  - authenticate()
  - authorize()
  - isEncrypted()
  - isTrusted()

## Communication API

- JSR 82 supports a low-level data transfer (Logic Link Control & Adaptation) and a stream communication interface (RFCOMM)
- L2CAP interface not commonly used
  - Need significant effort from user to handle the packets, fragmentation etc.,
- Stream communication API is used extensively in MIDP development
  - `btsp://hostname:[CN | UUID];parameters`
    - `btsp` indicates RFCOMM StreamConnection protocol
    - `hostname` is either localhost, or the Bluetooth address
    - `CN` is the port number to communicate
    - `UUID` is service identifier
    - `parameters` authenticate, authorize, and encrypt information
    - Eg: `btsp://hostname;authenticate=true;authorize=true;encrypt=true`

## Initialization Code Snippet

- Both the server and client initialize the Bluetooth stack at the beginning of using JSR 82 API
    - Servers set the discovery mode to : *GIAC (always visible), LIAC (visible for a short time), NOT\_DISCOVERABLE (never visible)*
    - Client retrieves the discovery agent
- ```
public void btInit_CLIENT() throws BluetoothStateException {
    // Retrieve the local device to get to the Bluetooth Manager
    localDevice = LocalDevice.getLocalDevice();
    // Clients retrieve the discovery agent
    discoveryAgent = localDevice.getDiscoveryAgent();
}

public void btInit_SERVER() throws BluetoothStateException {
    // Retrieve the local device to get to the Bluetooth Manager
    localDevice = LocalDevice.getLocalDevice();
    // Servers set the discoverable mode to GIAC
    localDevice.setDiscoverable(DiscoveryAgent.GIAC);
}
```

## Service Connection Code Snippet : Server

- Server Code for Setting up Bluetooth Services:
  - `String btService = "Chat";`
  - `// Bluetooth Service UUID of interest`
  - `private static final String stUUID = "??"; //Generated using uuidgen`
  - `private UUID btUUID = new UUID(stUUID, false);`
  - ...
  - `// Define the server connection URL`
  - `String connURL = "btsp://localhost:"+btUUID.toString()+";name="+btService;`
  - ...
  - `// Create a server connection (a notifier)`
  - `StreamConnectionNotifier scn = (StreamConnectionNotifier)`  
`Connector.open(connURL);`
  - ...
  - `// Wait for client requests`
  - `StreamConnection sc = scn.acceptAndOpen();`
  - `// Request received and accepted; get remote device id`
  - `RemoteDevice rd = RemoteDevice.getRemoteDevice(sc);`
  - `// Read data being sent from the remote device`
  - `DataInputStream dataIn = sc.openDataInputStream();`
  - `String s = dataIn.readUTF();`

### Service Connection Code Snippet : Client

- Client first does the discovery process and then uses the discoveredServices class to connect to the service

```
> // First get pointer to service record
> ServiceRecord sr =
  (ServiceRecord)discoveredServices.elementAt(i);
> // Use SR to get URL
> String connURL =
  sr.getConnectionURL(ServiceRecord.NOAUTHENTICATE
  _NOENCRYPT, false);
> // Open connection
> StreamConnection sc = (StreamConnection)
  Connector.open(connURL);
```



### Server Discovery : Client

- Client implements DiscoveryListener to look for nearby devices

```
> public class BtClient implements DiscoveryListener {
  - public void deviceDiscovered (..) { // Keep track of discovered remote
    devices
  - public void inquiryCompleted(int param) { // Trigger service search }
  - public void servicesDiscovered() { // Keep track of discovered services }
  - public void serviceSearchCompleted () { // dispatch thread to handle
    services }
  > try { look4Service =
    discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);}
  > catch(BluetoothStateException bse) { // Handle error }
```

- Things to Remember:

```
> Discovery is expensive; so cache discoveries locally
  - RemoteDevice[] localCache =
    discoveryAgent.retrieveDevices(DiscoveryAgent.CACHED);
```



### Service Discovery: Client

- Once a device is discovered you can then call searchServices() API

```
> RemoteDevice rd = discovered device;
> discoveryAgent.searchServices(attrs, uuids, rd, this);
```

- Then client calls the connection API and open dataoutput stream and writes to it.

```
> StreamConnection streamConnection =
  (StreamConnection) Connector.open(connectionURL);
> DataOutputStream dataout =
  streamConnection.openDataOutputStream();
> dataout.writeUTF(messageOut);
```

