Redes Inalámbricas

Programación Bluetooth

Pablo Serrano

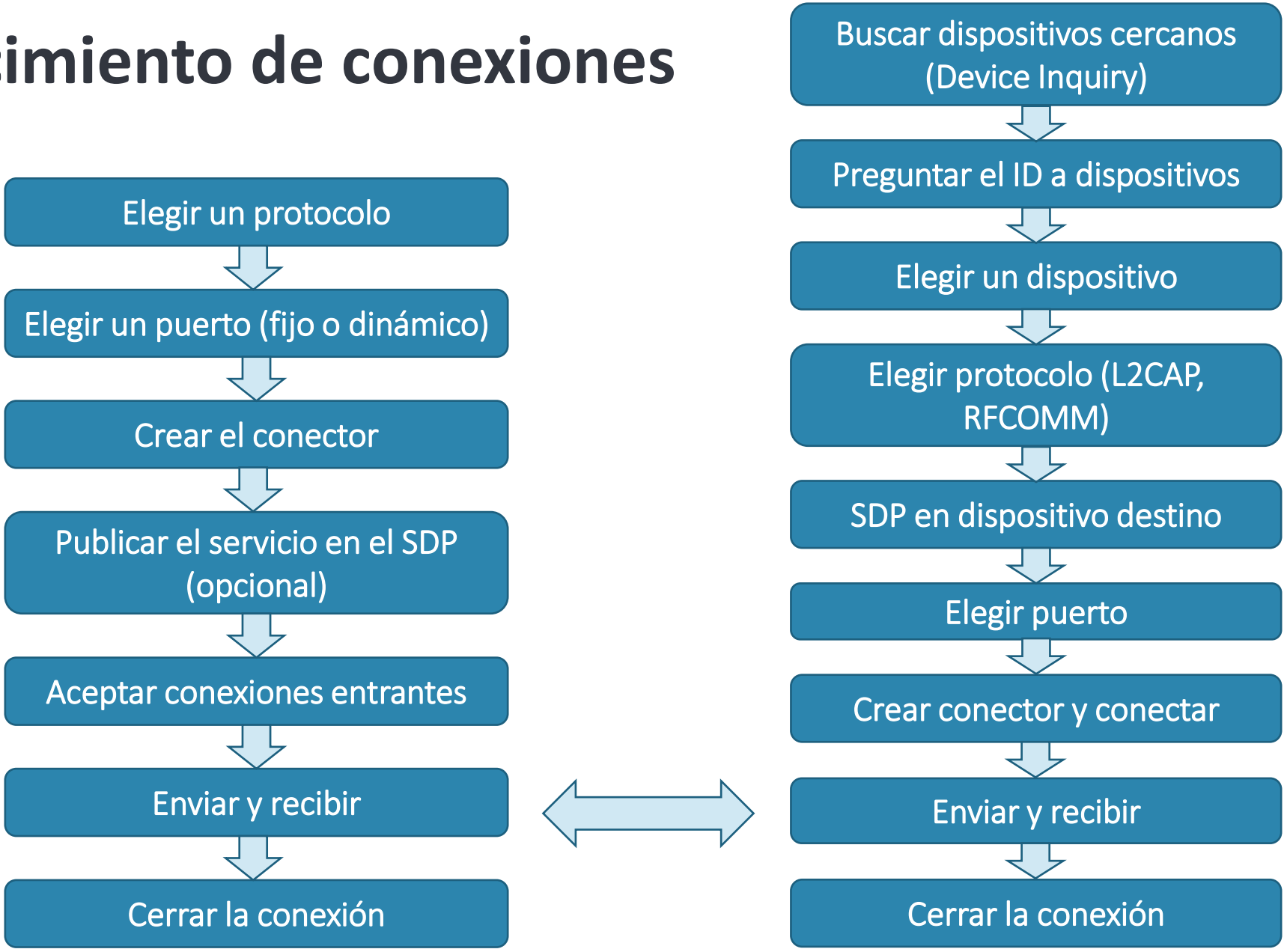
Grados de informática

Universidad de Málaga

Introducción

- ¿Cómo crear programas que conecten dos dispositivos Bluetooth?
- El proceso para establecer una conexión dependerá si el dispositivo en cuestión esta estableciendo una conexión de entrada (incoming) o de salida (outgoing)
  - Los dispositivos que inician una conexión de salida necesitan elegir un dispositivo destino (target) y un protocolo de transporte antes de establecer una conexión y transferir datos.
  - Los dispositivos que establecen una conexión de entrada necesitan elegir un protocolo de transporte, escuchar antes de aceptar una petición y transferir datos

Establecimiento de conexiones



Conexiones entrantes

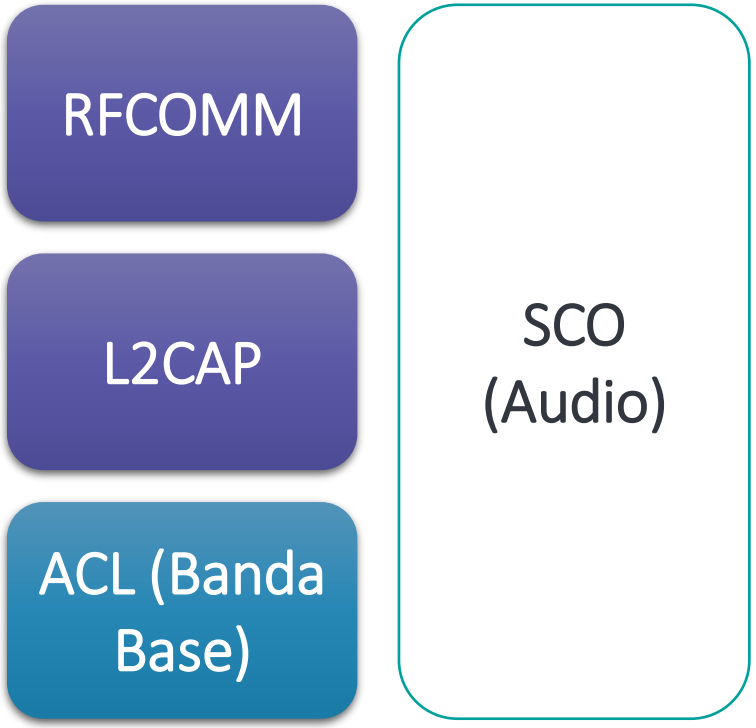
Conexiones salientes

Seleccionando un dispositivo

- Cada chip Bluetooth lleva incorporado de fábrica un identificador único de 48 bits, denominado dirección Bluetooth o dirección de dispositivo (Bluetooth address- device address)
- Es la única forma de direccionamiento en Bluetooth
- Es posible proporcionar un identificador “user-friendly”
  - Ej. “my phone”, “my car”, …
- Búsqueda de dispositivos en la proximidad: Device Inquiry

Conexiones. Protocolo de Transporte

- Bluetooth proporciona dos protocolos de transporte
  - RFCOMM (Radio Frequency Communications)
    - Protocolo orientado a flujo (similar a TCP)
  - L2CAP (Logical Link Control and Adaptation Protocol)
    - Protocolo orientado a paquete que puede configurase con varios niveles de fiabilidad
    - La configuración será compartida por TODAS las conexiones L2CAP a un mismo dispositivo
    - Sirve para encapsular conexiones RFCOMM



Puertos en Bluetooth

- Un puerto se utiliza para permitir que varias aplicaciones ejecuten conexiones en el mismo dispositivo
- En L2CAP los puertos se denominan Protocol Service Multiplexers (PSM) (Multiplexador de Servicios de Protocolos)
  - Toman valores IMPARES entre 1 y 32.767
- En RFCOMM los puertos se denominan canales (channels) estando disponibles del 1-30
- L2CAP reserva los puertos de 1-4095 para servicios estándar
  - SDP usa el puerto 1
  - Las conexiones RFCOMM se multiplexan por el puerto 3
  - L2CAP solo transporta datos, no voz y todos los puertos son fiables

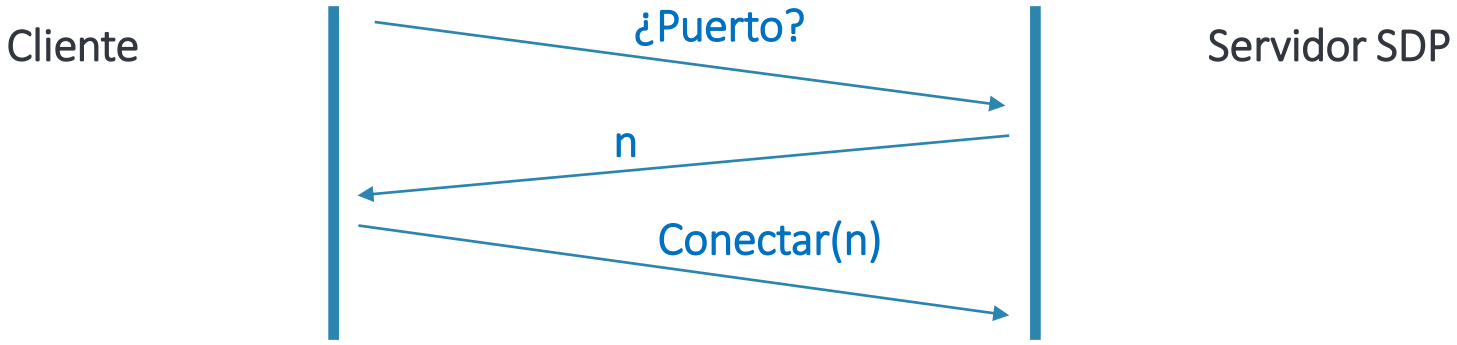
Puertos en Bluetooth

Protocolo	Término	Puertos reservados / conocidos	Puertos no reservados (libres)
RFCOMM	Canal	Ninguno	1-30
L2CAP	PSM	Impares 1-4095	Impares 4097-32765
SCO	n/a	n/a	n/a

- Dado que dos dispositivos pueden tener como máximo una conexión SCO entre ellos, no existe la noción de puerto en SCO

Protocolo de Descubrimiento de Servicios SDP

- Para conocer el puerto de conexión, se utiliza el SDP
- Cada dispositivo Bluetooth mantiene un servidor SDP
- Permite la asignación dinámica de puertos
  - Cuando un servicio arranca, puede elegir cualquier puerto arbitrario que no esté en uso.



Descripción de servicios

- Registro de Servicio - Service Record
  - Descripción de un servicio registrado por una aplicación en su servidor SDP y que posteriormente se transmitirá a los clientes durante el SDP
- Consiste en una lista de pares atributos /valor
  - Cada atributo es un entero de 16 bits
  - Cada valor puede ser de un tipo básico de datos (cadena de caracteres o entero)
- Los dos atributos más importantes son:
  - Service ID
  - Service Class ID List

Service Record	
ServiceID	0x1234-,,,
ServiceClassIDList	0xABCD-… 0x5678-…
ServiceName	Example Service
Protocol Description List	L2CAP RFCOMM Port 3

Atributos del SDP

- Service ID
- Service Class ID List
- Service Name
- Service Description
- Protocol Description List
- Profile Descriptor List
- Service Record Handle

Service ID

- Cada servicio tiene asignado un identificador único
- El cliente conoce este identificador
- El espacio de identificadores únicos válidos es mayor que el número de puertos
- Se usan UUID (Identificadores Universales Únicos) de 128 bits
- Cada desarrollador elige el UUID del servicio en tiempo de diseño y se asocia al registro del servicio
- Notación Estándar para un UUID: dígitos hexadecimales separados por guiones:
  - XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Service Class ID List

- Para servicios que proporcionan el mismo tipo de servicio se define un segundo UUID, denominado Service Class ID
- Dos programas diferentes pueden publicar el mismo Service Class ID
- Para las aplicaciones que ofrecen varios servicios, Bluetooth permite que cada servicio tenga una lista de clases de servicios proporcionados

UUIDs reservados

- Bluetooth tiene un conjunto de UUIDs reservados: SHORT UUIDs
- Permiten identificar clases de servicios predefinidos, protocolos de transporte y perfiles
- Los 96 bits más bajos de los UUIDs reservados tienen el mismo valor (UUID base), y se distinguen por los 16 o 32 bits superiores
- Para obtener el UUID de 128 bits a partir de un short UUID (16 o 32 bits):
  - 128_bit_UUID = 16_or_32_bit_number * 296 + Bluetooth_Base_UUID

UUIDs reservados

Servicio	UUID Reservado
SDP	0x0001
RFCOMM	0x0003
L2CAP	0x0100
SDP Service Class ID	0x1000
Serial Port Service Class ID	0x1101
Headset Service Class	0x1108

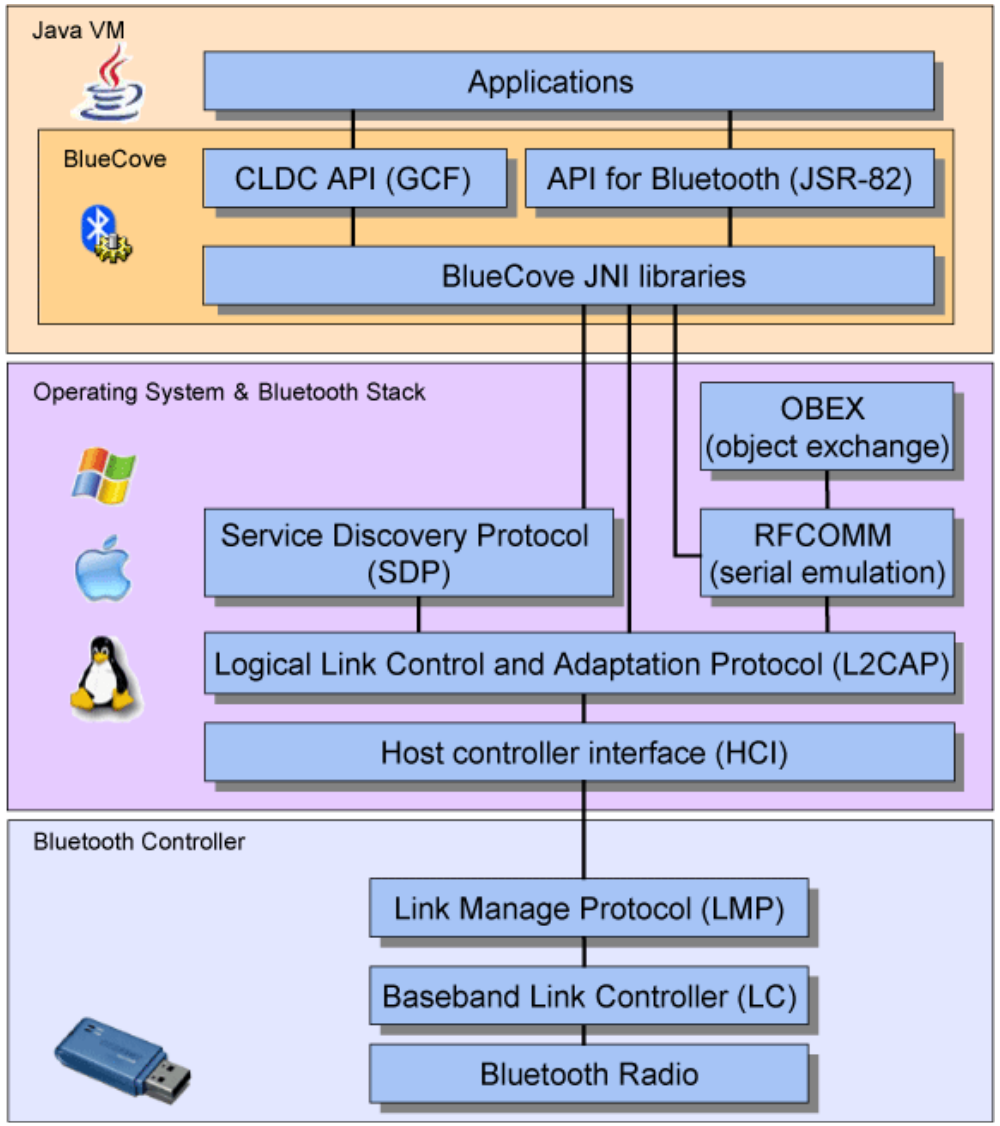
- Bluetooth Base UUID = 00000000-0000-1000-8000-00805F9B34FB

Java-JSR 82 BlueCove

Introducción a la programación Bluetooth

- Java JSR-82 es la especificación Java para la programación de aplicaciones Bluetooth
- BlueCove es colección de librerías Java para Bluetooth que implementan algunas interfaces de la especificación JSR-82
  - De código abierto (opensource) y de libre distribución.
  - Actualmente admite los siguientes perfiles
    - SDAP – Service Discovery Application Profile
    - RFCOMM – Serial Cable Emulation Protocol
    - L2CAP – Logical Link Control And Adaptation Protocol
    - OBEX – Generic Object Exchange Profile

Introducción a la programación Bluetooth



Librería BlueCove

- http://bluecove.org/
- Documentación: http://www.bluecove.org/bluecove/apidocs/
- Distribución de bluecove:
  - https://code.google.com/p/bluecove/
  - Incluir bluecove-2.1.0.jar en el proyecto Java
- Para 64 bits, última versión (2.1.1) (disponible en el Campus Virtual):
  - http://www.java2s.com/Code/Jar/b/Downloadbluecove211jar.htm
  - Incluir jar de BlueCove en el proyecto Java

BlueCove: Características generales

- Las clases javax.bluetooth.LocalDevice y javax.bluetooth.RemoteDevice representan un dispositivo (local o remoto, respectivamente)
  - Proporcionan información acerca de un dispositivo, incluyendo la dirección y el nombre del dispositivo
  - Identificación de dispositivos
    - Bluetooth address - device address
    - Display name - friendly name
- Obtener dispositivo local:

```
LocalDevice ld = LocalDevice.getLocalDevice();
```

- Obtener dispositivo remoto a partir de una conexión que hayamos establecido con dicho dispositivo:

```
RemoteDevice dev = RemoteDevice.getRemoteDevice(con);
```

BlueCove: Características generales

- Cada dispositivo Bluetooth tiene una dirección de 48 bits global única, denominada dirección bluetooth o dirección de dispositivo (Bluetooth address - device address)
  - Asignada por el fabricante del dispositivo
  - Utilizada en cualquier nivel o capa del proceso de comunicación Bluetooth.
  - Cadena de 12 caracteres
  - Método String getBluetoothAddress()
    - Ej.:
```
RemoteDevice rd;
rd.getBluetoothAddress();
```
- Cada dispositivo tiene un nombre de dispositivo “user-friendly” denominado nombre bluetooth (display name)
  - Asignado por el usuario (ej. “My phone”)
  - Son arbitrarios y pueden estar duplicados
  - Mostrado al usuario en lugar de la dirección bluetooth
  - Método String getFriendlyName(boolean alwaysAsk)
    - alwaysAsk, recomendado a false para evitar preguntar siempre

BlueCove: Descubrimiento de dispositivos

- Se realiza mediante un objeto denominado DiscoveryAgent
- El descubrimiento de dispositivos consta de tres pasos:
  - 1) Obtener una referencia al (único) objeto de la clase LocalDevice
    - Proporciona acceso y control al dispositivo bluetooth local

    ```
    LocalDevice ld = LocalDevice.getLocalDevice();
    ```
  - 2) Obtener una referencia al DiscoveryAgent del LocalDevice

    ```
    DiscoveryAgent da = ld.getDiscoveryagent();
    ```
  - 3) Utilizar el DiscoveryAgent para comenzar descubrimiento de dispositivos a través del método

    ```
    startInquiry(int accessCode, DiscoveryListener listener);
    ```

BlueCove: Inquiry

```
startInquiry(int accessCode, DiscoveryListener listener);
```

- accessCode
  - Dos códigos de acceso:
    - GIAC - General Inquiry access Code - (General – buscamos dispositivos visibles para todos) -> Usaremos este
    - LIAC – Limited Inquiry access Code) – (Limitado – visibles sólo para los que conocemos)
- DiscoveryListener
  - Se trata de una interfaz que debemos implementar para pasarla como argumento a la función. Se usa tanto para el descubrimiento de dispositivos como de servicios.
- Uso:

```
MyDiscoveryListener lstnr = MyDiscoveryListener();
da.startInquiy(Discoveryagent.GIAC, lstnr);
```

BlueCove: DiscoveryListener

- Inteface javax.bluetooth.DiscoveryListener
- Es necesario implementar los siguientes métodos de esta interfaz:
  - deviceDiscovered: será el método que se ejecute cuando se encuentre un nuevo dispositivo durante el descubrimiento de dispositivos
  - inquiryCompleted: se ejecuta al finalizar la búsqueda
  - servicesDiscovered: es invocado cuando se encuentran servicio(s) durante la búsqueda de servicios
  - serviceSearchCompleted: es invocado al completarse la búsqueda de dispositivos o se finaliza debido a un error
- Los procedimientos de búsqueda se realizan de forma asíncrona, de forma que nuestro programa no se quede esperando mientras se realizan

Ejecución concurrente y asíncrona

- Es recomendable usar semáforos o monitores para controlar el fin del proceso de descubrimiento
- 1) En el proceso principal paramos el hilo de ejecución hasta que la búsqueda no se complete. Ejemplo:

```
final Object inquiryCompletedEvent = new Object();
public static void main(String[] args){
    …
    da.startInquiry(DiscoveryAgent.GIAC, discoveryListener);
    synchronized(inquiryCompletedEvent) {
    try { inquiryCompletedEvent.wait(); }
    catch ( Exception e ) {}
    }
}
```

  - El objeto de espera podría sustituirse por un semáforo o por la clase del main (this) si implementa DiscoveryListener

Ejecución concurrente y asíncrona

- 2) En el DiscoveryListener avisamos al proceso principal cuando la búsqueda de dispositivos o servicios termine
  - Para ello, modificamos el cuerpo de las funciones inquiryCompleted() y serviceSearchCompleted(), respectivamente. Ejemplo:

```
synchronized(inquiryCompletedEvent ) {
try { inquiryCompletedEvent.notifyAll();
} catch ( Exception e ) {}
}
```

Descubrimiento de servicios - SDP

- La búsqueda de servicios en un dispositivo se realiza con el método searchServices del DiscoveryAgent. Posteriormente, hará uso de las mismas clases que el descubrimiento de dispositivos.
- En JSR-82 no hay funciones especificas para la búsqueda de servicios de dispositivos en las cercanías, por lo que debe realizarse específicamente buscando uno por uno
- Las clases que representan los servicios y su descripción son:
  - ServiceRecord: registro de servicio, contiene DataElements
  - DataElement: par atributo/valor
  - UUID: identifica a clases de servicios o atributos determinados. Son enteros sin signo de 16-bit (UUID)
- Hay un conjunto de atributos de servicio que siempre se recuperan (o incluyen) en el ServiceRecord del servicio
  - Service Record Handle, Service Class ID List, Service Record State, Service ID, Protocol Descriptor List

Descubrimiento de servicios - SDP

- A partir de una lista de dispositivos detectados se itera para cada uno de los dispositivos detectados, y se buscan los servicios de una determinada clase, por ejemplo, los que ofrece públicamente (cuyo identificador UUID de clase de servicio es 0x1002)
- El resultado de la búsqueda (en el método servicesDiscovered de la interfaz DiscoveryListener) devuelve un array de ServiceRecords con todos los servicios descubiertos
- La clase ServiceRecord contiene objetos de clase DataElement que informan sobre un servicio descubierto

`getConnectionURL`

- Devuelve una cadena conteniendo la dirección Bluetooth y el puerto para establecer posteriormente la conexión

`getAttributeValue`

- Accede a los elementos almacenado en una instancia de la clase ServiceRecord

Ejemplo descubrimiento de servicios: obtención de nombre y URL

```
UUID uuids[] = new UUID[1];
uuids[0]= new UUID (0x1002);
int attridset[] = new int[1];
attridset[0] = SERVICE_NAME_ATTRID; //0x0100
da.searchServices(attridset, uuids, remotedevice,lstnr);
……
public void servicesDiscovered(int transID, ServiceRecord[] rec) {
    for (int i=0; i<rec.length;i++) {
        DataElement d = rec[i].getAttributeValue(SERVICE_NAME_ATTRID );
        if (d!=null) System.out.println((String)d.getValue());
        else System.out.println("Unnamed service");
        System.out.println(rec[i].getConnectionURL(
                ServiceRecord.NOAUTHENTICATE_NOENCRYPT,false));
        }
}
```

Programación de Sockets Bluetooth

Programación con Sockets RFCOMM

- Establecimiento de conexiones entrantes (Servidor)
  - 1) Definir una URL que especifica datos necesarios para
    - 1. Escuchar peticiones
    - 2. Anunciar el servicio (Definir el ServiceRecord)
    - Estructura de la URL para RFCOMM
      - **“btspp://” + “dirección Bluetooth del adaptador local”+”:” + “Service Class ID” + parámetros opcionales**
      - Hasta 5 parámetros opcionales de la forma “attribute=value”:
        - Name: Nombre de servicio
        - Authenticate (false): Si true, los dispositivos clientes deben autenticarse. Default: false
        - Encrypt (false): Si true, la conexión está encriptada
        - Authorize (false): Si true se requiere autorización (base de datos o usuario) antes de aceptar peticiones entrantes.
        - Master (false): El adaptador local es maestro en la pico red (piconet) asociada a las conexiones entrantes
  - 2) La clase StreamConnectionNotifier es equivalente a socket de servidor y es necesaria para aceptar peticiones de conexión entrantes

```
StreamConnectionNotifier service = (StreamConnectionNotifier)
Connector.open(url);
```

Programación con Sockets RFCOMM

- 3) El servicio espera y acepta peticiones con el método acceptAndOpen sobre este objeto

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

- 4) Cuando se acepta una petición devuelve un objeto de la clase StreamConnection, que se utiliza para intercambiar información con el dispositivo Bluetooth remoto.

- Establecimiento de conexiones salientes (Cliente)
  - 1) Construir una URL del servicio al que se quiere conectar
    - Manual o extraída del ServiceRecord del servicio
    - Tres parámetros opcionales:
      - Master (false). El dispositivo cliente se convierte en maestro de la piconet
      - Encrypt
      - authenticate
  - 2) Utilizar la URL para establecer una conexión a través del método (estático) open de la clase Connector.

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

Uso de un socket RFCOMM conectado

- Hay que obtener instancias de las clases OutputStream e InputStream para enviar y recibir datos a través de la conexión respectivamente
  - OutputStream os = con.openOutputStream();
  - InputStream is = con.openInputStream();
- Para enviar datos se pasa un array de bytes al método write del objeto OutputStream
  - Devuelve los bytes enviados

```
String mensaje = "hola";
int bytes_sent = os.write(mensaje.getBytes());
```

- Para recibir datos se pasa un array de bytes vacio al método read del objeto InputStream
  - Devuelve el número del nº de bytes leídos con éxito

```
byte read_buffer[] = new byte[80];
int bytes_received = is.read (read_buffer);
```

- La conexión se cierra con el método close:

```
con.close();
```

Algunas consideraciones

- Existen varias implementaciones de stacks (pila de protocolos) Bluetooth. BlueCove es dependiente de la pila de protocolos que tenga implementada el sistema, limitando o ampliando así las opciones propias de JSR-82.
- BlueCove es compatible con la Bluetooth Stack de Microsoft (winsock), que es la que garantizará el correcto funcionamiento de la aplicación, ya que es el dispositivo bluetooth utilizado es compatible con esta pila.
- Limitaciones en la pila de protocolo de Windows:
  - Microsoft Bluetooth Stack solo soporta conexiones RFCOMM.
  - No soporta L2CAP
  - DiscoveryListener.deviceDiscovered() será llamado para dispositivos emparejados con la pila de protocolos de Microsoft independientemente de que el dispositivo esté apagado o encendido.
  - LocalDevice.setDiscoverable(NOT_DISCOVERABLE) no cambiará el estado del dispositivo si está en modo DISCOVERABLE (puede ser visto por otros dispositivos). Esta opción únicamente se podrá modificar desde la configuración del propio dispositivo bluetooth y no desde el propio código.
  - No está implementada la autenticación, autorización y encriptación, provista normalmente por el objeto RemoteDevice del JSR-82.
- Limitaciones de la API BlueCove en: https://code.google.com/p/bluecove/wiki/stacks