ANEXO C. MODULOS DE LA APLICACIÓN

TABLA DE CONTENIDO

ANEXO C MODULOS DE LA APLICACIÓN

C.1 ASPECTOS GENERALES DE UNA APLICACIÓN BLUETOOTH QUE EMPLEA LA API JSR-82

La estructura de una aplicación Blueooth consiste de 5 partes: Inicialización del stack de Protocolos, Gestión de dispositivos, Descubrimiento del dispositivo, Descubrimiento del Servicio y el Establecimiento de una comunicación.

C.1.1 Inicialización del Snack

El stack de protocolos de Bluetooth es el directo responsable del control de los dispositivos, y es por esta razón que antes de hacer cualquier cosa se debe inicializar. El proceso de inicialización comprende varios pasos con los cuales se debe obtener el dispositivo que se encuentre listo para establecer una comunicación inalámbrica. Desafortunadamente la especificación Bluetooth permite que la inicialización del *stack* sea diferente de acuerdo a cada fabricante por ello en algunos dispositivos puede ser una Interfaz de Usuario, mientras que en otros pueden ser unas variables preestablecidas y que no las puede modificar el usuario. A continuación se muestra una posible solución para la inicialización del *Stack*, sin embargo es de anotar que cada fabricante lo puede hacer de una forma diferente:

```
...
// Fijar el numero del puerto
BCC.setPortNumber("COM1");
// Fijar la rapidez
BCC.setBaudRate(50000);
// Fijar el modo conectable
BCC.setConnectable(true);
// Fijar el modo de descubrimiento
// para un código de Acceso a Indagación limitado.
BCC.setDiscoverable(DiscoveryAgent.LIAC);
...
```

C.1.2 Gestión de Dispositivos

Las clases *LocalDevice* y *RemoteDevice*, proporcionan capacidades para la gestión del dispositivo definida en el Perfil de Acceso Genérico. *LocalDevice* depende de la clase *javax.bluetooth.DeviceClass* para recuperar el tipo de dispositivo y la clase de servicio que ofrece. La clase *RemoteDevice* representa un dispositivo remoto (dispositivo que se encuentra dentro del rango de cobertura) y proporciona los métodos que permiten recuperar información relacionada

con el dispositivo incluyendo su nombre y dirección Bluetooth. En el siguiente código se puede ver como recuperar la información del dispositivo local:

```
...
// Recupera un objeto dispositivo local Bluetooth
LocalDevice local = LocalDevice.getLocalDevice();
// Recupera la dirección Bluetooth del dispositivo local
String address = local.getBluetoothAddress();
// Recupera el nombre del dispositivo local
String name = local.getFriendlyName();
...
```

Para el caso del dispositivo remoto se tiene:

```
...
// Recupera el dispositivo que esta al otro lado de la conexión
// del Perfil de Puerto Serial, L2CAP u OBEX sobre RFCOMM.
RemoteDevice remote =
RemoteDevice.getRemoteDevice(
javax.microedition.io.Connection c);
// Recupera la dirección Bluetooth del dispositivo remoto
String remoteAddress = remote.getBluetoothAddress();
// Recupera el nombre del dispositivo remoto
String remoteName = local.getFriendlyName(true);
...
```

La clase *RemoteDevice* también proporciona métodos para autenticar, autorizar o encriptar los datos transmitidos entre los dispositivos local y remoto.

C.1.3 Descubrimiento del Dispositivo

Debido a que los dispositivos inalámbricos están en continuo movimiento, es necesario tener algún mecanismo que le permita al equipo móvil encontrar otros dispositivos y obtener acceso a sus capacidades. La clase *DiscoveryAgent* y la interfaz *DiscoveryListener* ofrecen los servicios de descubrimiento necesarios.

Un dispositivo Bluetooth puede utilizar un objeto *DiscoveryAgent* el cual se obtiene de una lista de dispositivos accesibles, para ello puede emplear alguna de las siguientes opciones:

- El método *DiscoveryAgent.startInquiry* pone al dispositivo en modo indagación, para tomar ventaja de este modo, la aplicación especifica un evento “listener”, el cual responderá a los eventos relacionados con indagaciones. Así pues el *DiscoveryListener.deviceDiscovered* se llama cada vez que a un dispositivo se le haga una indagación, finalmente cuando la indagación se completa o se cancela, se invoca al método *DiscoveryListener.inquiryCompleted.*
- Si el dispositivo no desea esperar para descubrir cuales dispositivos pueden ser descubiertos, entonces puede emplear el método *DiscoveryAgent.retrieveDevices* para recuperar una lista ya existente. Dependiendo del parámetro que se le pase, este método

devuelve una lista con los dispositivos que se encontraron en la indagación anterior o una lista de dispositivos “preconocidos” los cuales se conectan a menudo.

A continuación se explica en más detalle estos procedimientos:

```
...
// Recupera el agente descubrimiento
DiscoveryAgent agent = local.getDiscoveryAgent();
// Se establece el dispositivo en modo indagación
boolean complete = agent.startInquiry();
...
...
// Recupera el agente descubrimiento
DiscoveryAgent agent = local.getDiscoveryAgent();
// Devuelve un arreglo con la lista de dispositivos “Preconocidos”
RemoteDevice[] devices =
agent.retrieveDevices(DiscoveryAgent.PREKNOWN);
...
...
// Recupera el agente descubrimiento
DiscoveryAgent agent = local.getDiscoveryAgent();
// Devuelve un arreglo con la lista de
// dispositivos encontrados en una indagación previa
RemoteDevice[] devices =
agent.retrieveDevices(DiscoveryAgent.CACHED);
...
```

C.1.4 Descubrimiento del Servicio

Una vez que el dispositivo local ha sido descubierto al menos una vez por el dispositivo remoto, puede iniciar la búsqueda de los servicios disponibles. El descubrimiento del servicio es bastante parecido al descubrimiento de dispositivo, por esta razón el *DiscoveryAgent* también proporciona métodos que permiten descubrir servicios en un dispositivo servidor Bluetooth y además inicializar las transacciones relacionadas con el descubrimiento del servicio. Cabe aclarar que para esta API se ofrecen mecanismos para la búsqueda de servicios en dispositivos remotos pero no para la búsqueda de servicios en dispositivos locales.

C.1.5 Registro del Servicio

Antes de que algún servicio pueda ser descubierto es necesario que dicho servicio sea registrado en un dispositivo *servidor Bluetooth*, este dispositivo servidor tiene las siguientes responsabilidades:

- Crear un registro de servicio y además describirlo.
- Adicionar dicho registro de servicio a la base de datos de descubrimiento del servicio en el servidor, para que el registro sea visible y este disponible a los clientes.
- Registrar las medidas de seguridad Bluetooth asociadas con el servicio, para así fortalecer las conexiones con los clientes.
- Aceptar las conexiones con los clientes.

- Actualizar el registro de servicio en la base de datos cada vez que cambien los atributos del servicio.
- Eliminar o deshabilitar el registro de servicio de la base de datos, cuando el servicio ya no se encuentre disponible.

Los siguientes fragmentos de código muestran como se hace el registro del servicio:

a. Crear un Nuevo registro de servicio que represente el servicio, se invoca el método *Connector.open* con un argumento URL de conexión con el servidor y el resultado se le pasa a un *StreamConnectionNotifier* el cual representa el servicio:

```
...
StreamConnectionNotifier service =
(StreamConnectionNotifier) Connector.open("alguna URL");
```

b. Obtener el registro de servicio creado por el dispositivo servidor:

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

c. Anunciar que el servicio esta listo para aceptar conexiones con los clientes:

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

El acceptAndOpen permanece bloqueado hasta que un cliente se conecte.

d. Cuando el servidor este listo para terminar, se cierra la conexión y se elimina el registro de servicio:

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

C.1.6 Establecimiento de la comunicación

Para que un dispositivo local pueda utilizar un servicio sobre un dispositivo remoto es necesario que los dos dispositivos compartan un protocolo de comunicación en común, una vez se ha realizado esto, las aplicaciones pueden acceder a una amplia variedad de servicios Bluetooth; Esta API ofrece mecanismos de conexión a cualquier servicio que utilice RFCOMM, L2CAP u OBEX como protocolo de comunicación.

C.1.7 Perfil de Puerto Serial

El protocolo RFCOMM, emula una conexión de Puerto serial RS-232. El Perfil de Puerto Serial (SPP) facilita la comunicación entre dispositivos Bluetooth proporcionando una interfaz con el protocolo RFCOMM. A continuación se pueden ver algunas capacidades y limitaciones del SPP:

- Dos dispositivos solo pueden compartir una sesión RFCOMM al tiempo.
- En la misma sesión se pueden multiplexar más de 60 conexiones seriales lógicas.
- Un solo dispositivo Bluetooth puede tener al menos 30 servicios RFCOMM activos.

- Un dispositivo puede soportar solo una conexión cliente para un servicio dado.

Para una comunicación entre un cliente y un servidor usando el Perfil de Puerto Serial, cada uno de ellos debe seguir unos pasos muy simples.
El siguiente código muestra lo que debe hacer el servidor:

a. Construir una URL que indique como conectarse al servicio y almacenarla en el registro de servicio. Dicha URL puede parecer algo como *btspp://102030405060740A1B1C1D1E100:5* Aquí se dice que el cliente puede establecer una conexión al servicio que esta identificado con el canal servidor 5 en un dispositivo cuya dirección es *102030405060740A1B1C1D1E100*.
b. Hacer que el registro de servicio se encuentre disponible para el cliente.
c. Aceptar la conexión con el cliente.
d. Enviar y recibir datos hacia y desde el cliente.

```
...
// Se asume que el UID del servicio se ha recibido
String serviceURL =
"btspp://localhost:"+serviceUID.toString());
// más explicitamente:
String ServiceURL =
"btspp://localhost:10203040607040A1B1C1DE100;name=SPP
Server1";
try {
// crea una conexión servidora
StreamConnectionNotifier notifier =
(StreamConnectionNotifier) Connector.open(serviceURL);
// acepta las conexiones con los clientes
StreamConnection connection = notifier.acceptAndOpen();
// se prepara para enviar o recibir datos
byte buffer[] = new byte[100];
String msg = "hello there, client";
InputStream is = connection.openInputStream();
OutputStream os = connection.openOutputStream();
// envía datos hacia el cliente
os.write(msg.getBytes());
// lee los datos del cliente
is.read(buffer);
connection.close();
} catch(IOException e) {
e.printStackTrace();
}
...
```

En el otro extremo, para que el cliente pueda establecer una conexión RFCOMM con el servidor debe:

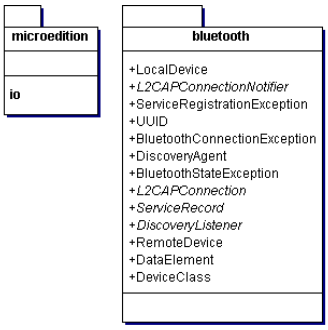
a. Iniciar el descubrimiento del servicio para recibir el registro del servicio.
b. Construir una URL de conexión usando el registro de servicio.
c. Abrir una conexión con el servidor.
d. Enviar y recibir datos hacia y desde el servidor.

```
...
// Se asume que ya se tiene el registro de servicio
// el registro se usa para recibir la URL de conexión
String url =
record.getConnectionURL(
record.NOAUTHENTICATE_NOENCRYPT, false);
// Abrir una conexión con el servidor
StreamConnection connection =
(StreamConnection) Connector.open(url);
// Enviar y recibir datos
try {
byte buffer[] = new byte[100];
String msg = "hola aquí el servidor";
InputStream is = connection.openInputStream();
OutputStream os = connection.openOutputStream();
// envía datos hacia el servidor
os.write(msg.getBytes);
// lee datos desde el servidor
is.read(buffer);
connection.close();
} catch(IOException e) {
e.printStackTrace();
}
...
```

C.2 *PAQUETES, CLASES Y METODOS*

A continuación se hace una breve descripción de las clases definidas para paquetes involucrados en el desarrollo de la aplicación, entre ellos se encuentran: javax.bluetooth, javax.microedition.io y org.javabluetooth. El diagrama de clases ofrece una percepción de todos los métodos definidos para cada una de las clases, sin embrago, en este documento solo se mencionan puesto que hacer una descripción de cada uno de ellos es muy extenso, para una mejor comprensión es recomendable leer los apéndices de la API JSR-82.

C.2.1 PAQUETE javax



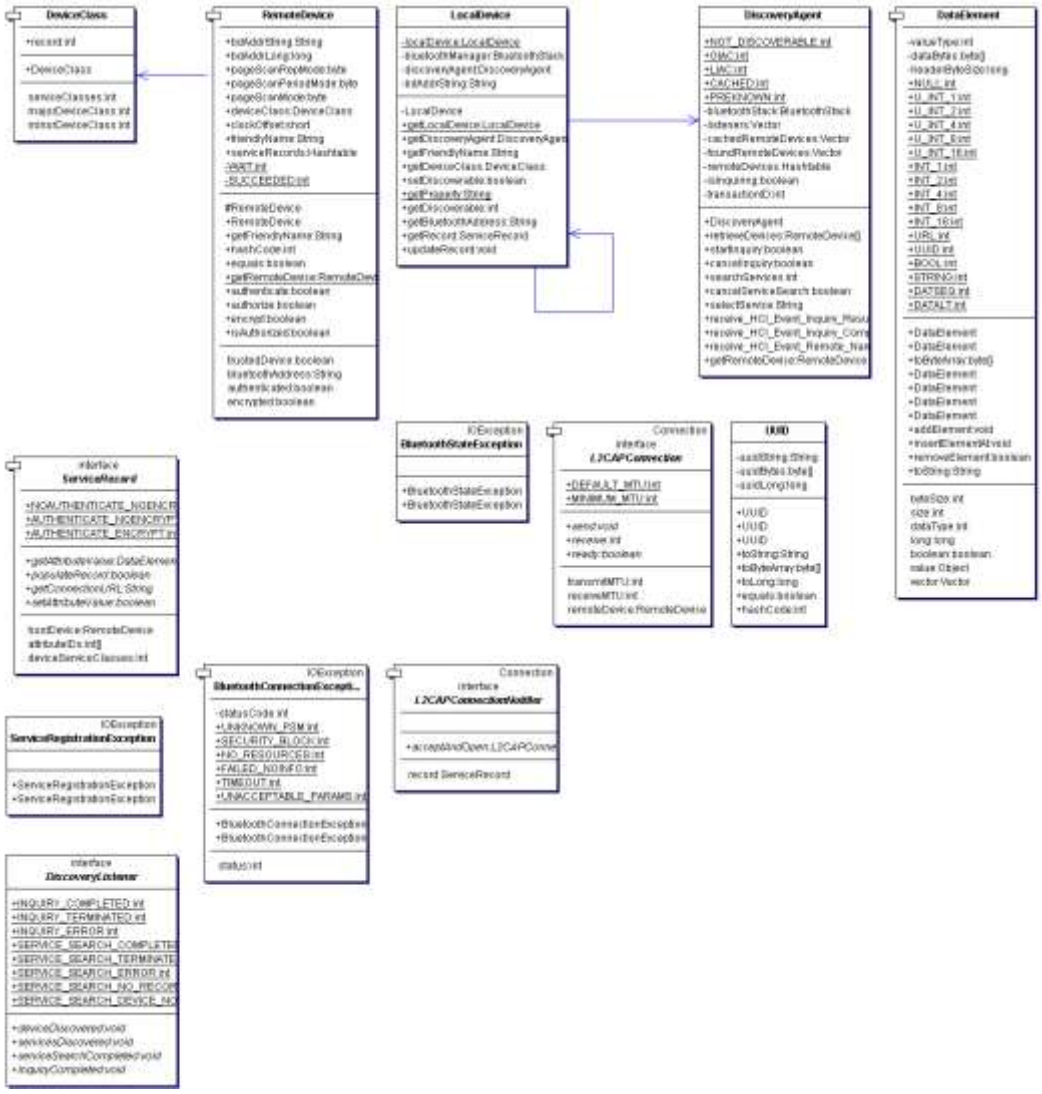
C.2.2 PAQUETE javax.bluetooth

INTERFACES	
DiscoveryListener	Esta Interfaz permite que una aplicación reciba eventos relacionados tanto con el descubrimiento de dispositivos como de servicios.
L2CAPConnection	Esta Interfaz representa un canal L2CAP orientado a la conexión.
L2CAPConnectionNotifier	Proporciona un notificador de conexión L2CAP.
ServiceRecord	Describe las características de un servicio.

CLASES	
BluetoothConnectionException	Esta excepción se ejecuta cuando no se puede establecer satisfactoriamente una conexión, ya sea L2CAP, RFCOMM u OBEX sobre RFCOMM.
BluetoothStateException	Esta excepción ocurre cuando el sistema se encuentra en un estado en el cual no puede responder a una solicitud hecha.
DataElement	Define el tipo de dato que puede tener el valor del atributo del servicio.

DeviceClass	Representa la clase de dispositivo, la cual esta definida por la especificación de Bluetooth.
DiscoveryAgent	Proporciona los métodos para modificar tanto el descubrimiento del dispositivo como del servicio.
LocalDevice	Define las funciones básicas del gestor Bluetooth, el gestor brinda acceso y control del dispositivo local.
RemoteDevice	Representa un dispositivo remoto y brinda información básica relacionada con el nombre y la dirección Bluetooth de dicho dispositivo.
ServiceRegistrationException	Esta excepción se ejecuta cuando se presenta una falla al adicionar o modificar un registro de servicio en la Base de Datos del Descubrimiento del Servicio - SDDB.
UUID	Define un Identificador Único Universal.

Diagrama de clases:

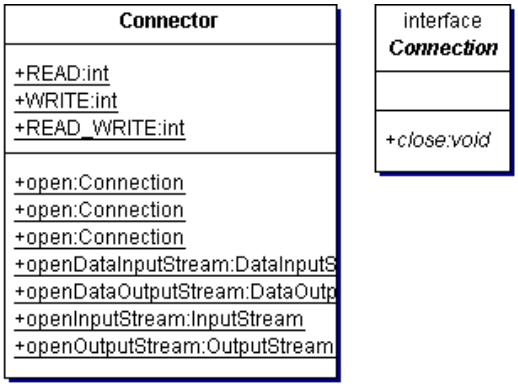


C.2.3 PAQUETE javax.micoedition.io

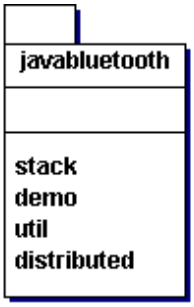
INTERFACES	
Connection	Esta Interfaz se emplea por compatibilidad con la maquina virtual.

CLASES	
Connector	Esta clase ofrece las características mínimas requeridas por el paquete javax.bluetooth, además ofrece todos los métodos que permiten crear objetos de conexión.

Diagrama de Clases:



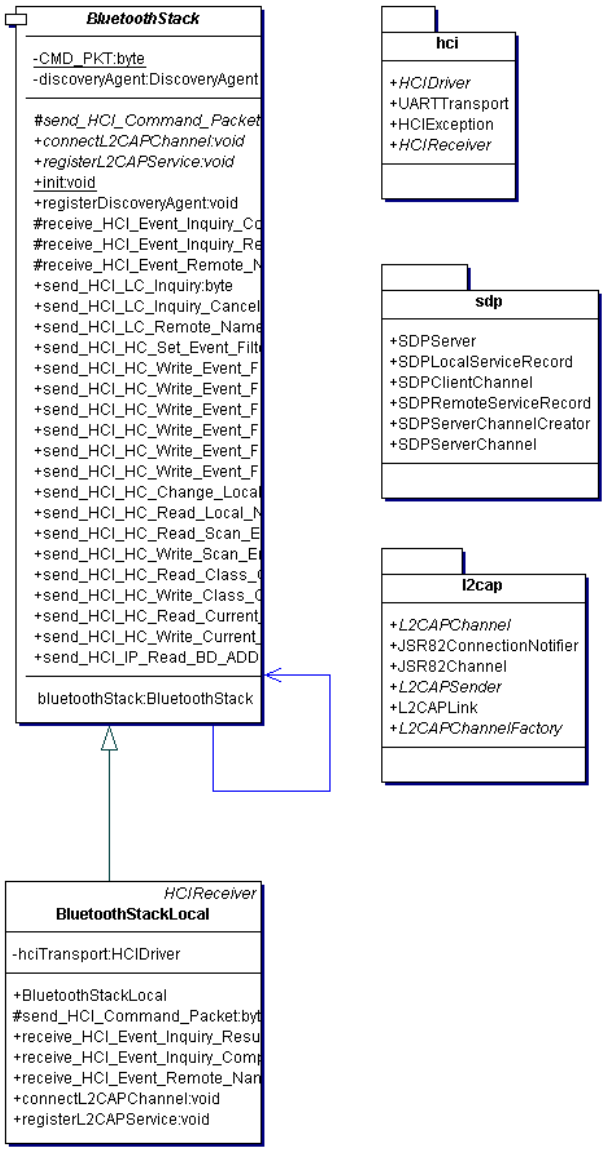
C.2.4 PAQUETE org.javabluetooth



C.2.5 PAQUETE org.javabluetooth.stack

CLASES	
BluetoothStack	Esta clase proporciona los niveles superiores del stack de protocolos de Bluetooth.
BluetoothStackLocal	Esta clase es una pequeña aplicación local del stack de protocolos de Bluetooth.

Diagrama de clases:



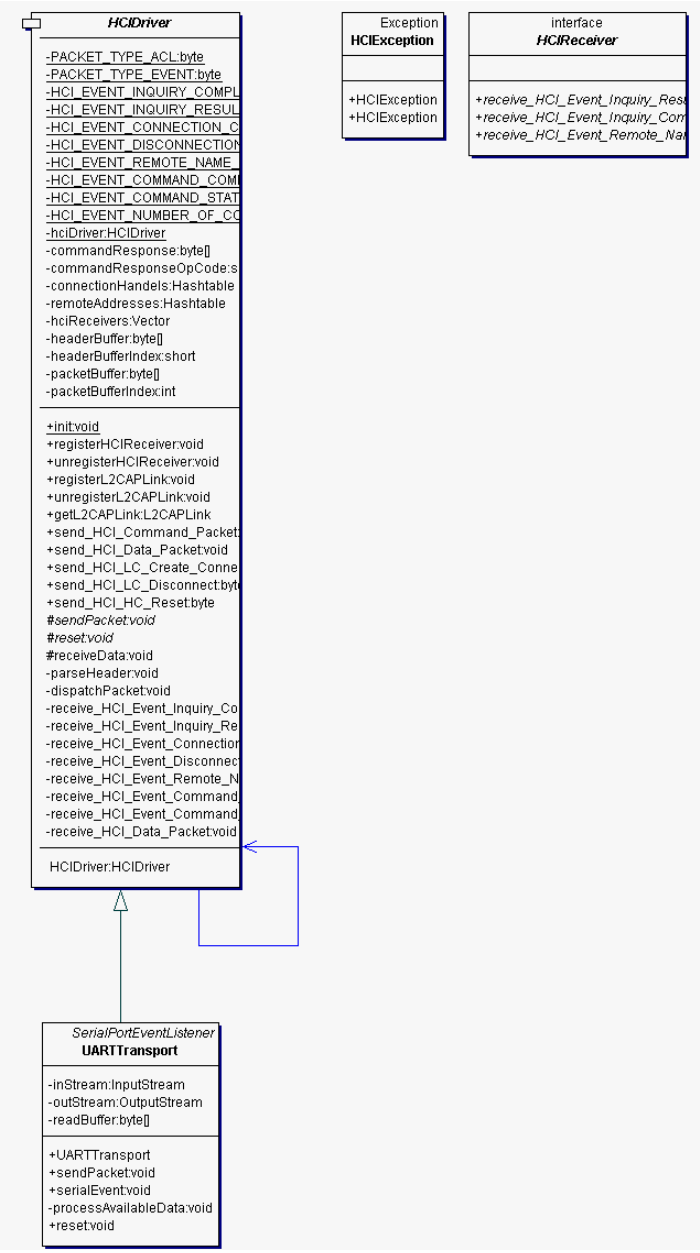
C.2.6 PAQUETE org.javabluetooth.stack.hci

INTERFACES	
HCIReceiver	Esta Interfaz describe los eventos que deben soportar las clases que la implementan.

CLASES	
HCIDriver	Esta clase es implementada por los drivers para transporte HCI.

HCIException	Esta excepción indica algún error sobre la capa HCI.
UARTTransport	Esta clase es la aplicación HCITransport del Protocolo Serial UART.

Diagrama de Clases:



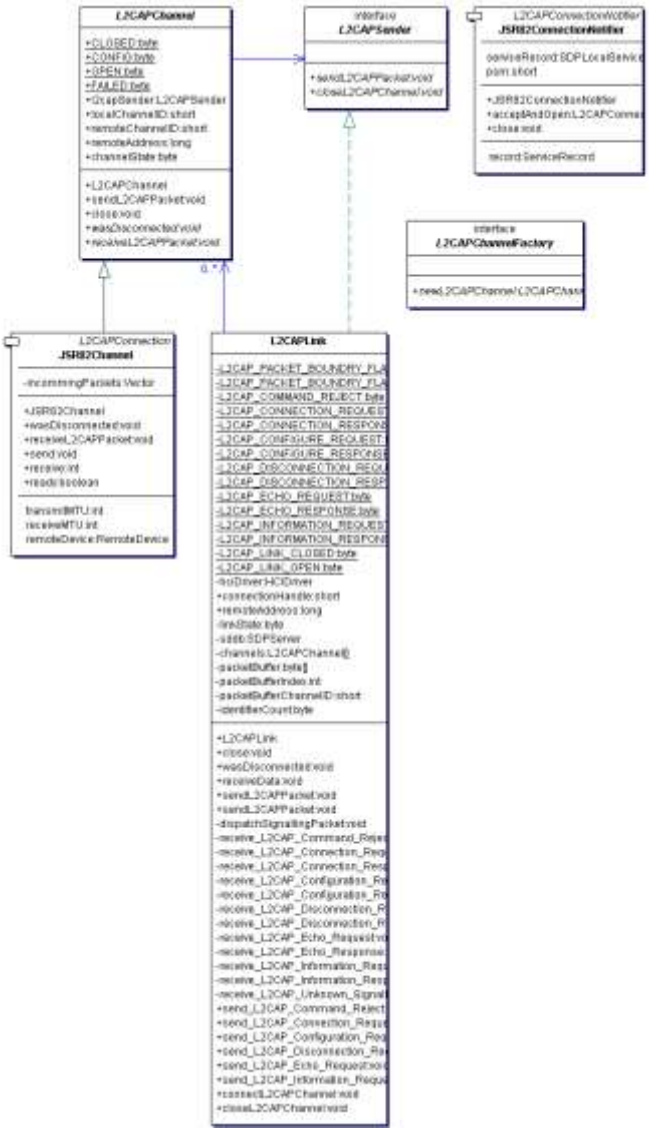
C.2.7 PAQUETE org.javabluetooth.stack.l2cap

INTERFACES	
L2CAPChannelFactory	Esta Interfaz es implementada por las clases *Factory* que crean los canales L2CAP.

L2CAPSender	Esta interfaz es implementada por las clases que gestionan los canales L2CAP.

CLASES	
JSR82Channel	Esta clase se utiliza en el método *Connector.open(String url)* para encapsular el canal L2CAP sobre una conexión L2CAP.
JSR82ConnectionNotifier	Esta clase se utiliza en el método *Connector.open(String url)* para crear el registro del servicio y abrir un canal JSR82.
L2CAPChannel	Esta clase representa un canal L2CAP entre dos dispositivos Bluetooth.
L2CAPLink	Esta clase representa un enlace entre dos dispositivos Bluetooth.

Diagrama de clases:

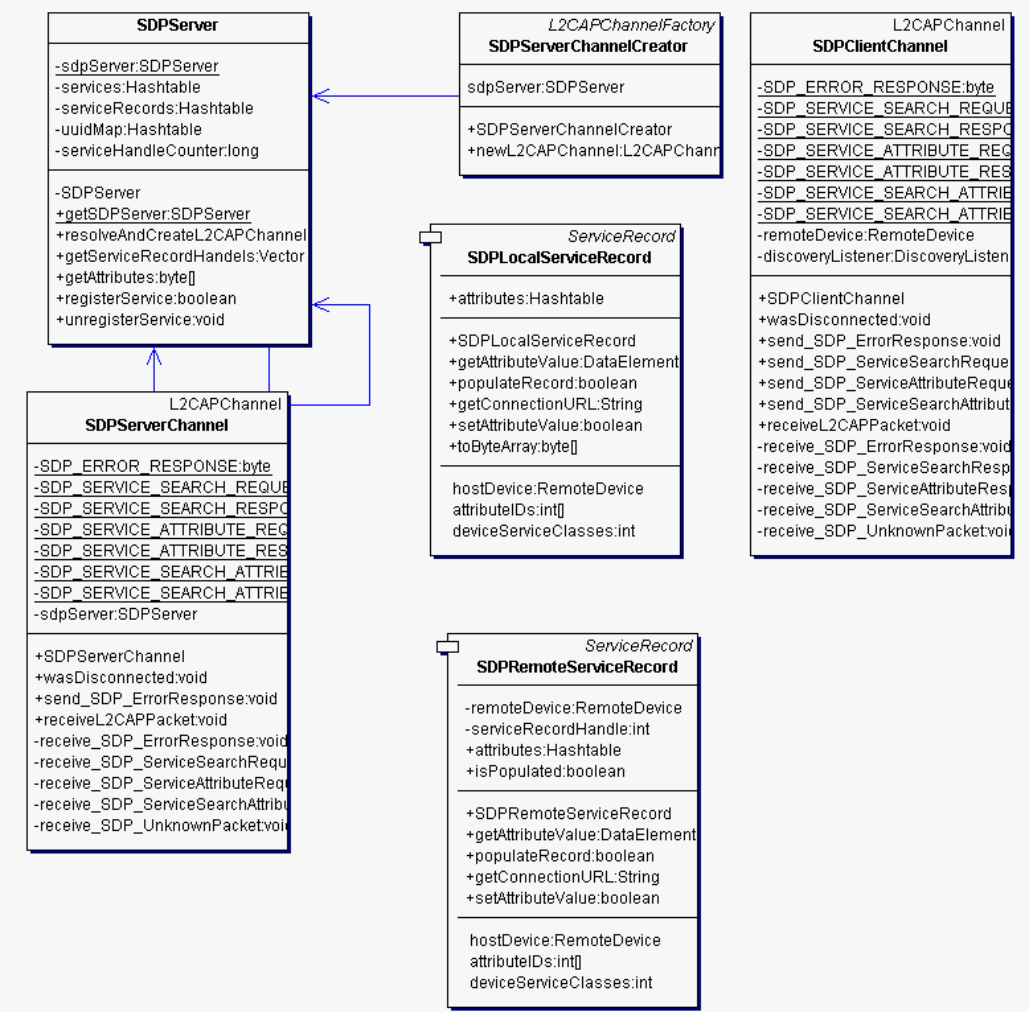


C.2.8 PAQUETE org.javabluetooth.stack.sdp

CLASES	
SDPClientChannel	Implementa el canal L2CAP para el Protocolo de Descubrimiento del Servicio.
SDPLocalServiceRecord	Esta clase implementa la interfaz javax.bluetooth.ServiceRecord y ofrece una representación de un Registro de Servicio tal y como se almacena en Base de datos local de Descubrimiento del Servicio.
SDPRemoteServiceRecord	Esta clase implementa la interfaz javax.bluetooth.ServiceRecord que se utiliza para

	representar un servicio encontrado en un dispositivo Bluetooth remoto y además contiene tanto los atributos del servicio como las instrucciones de cómo conectarse a dicho servicio.
SDPServer	Gestiona la base de datos local del descubrimiento del servicio.
SDPServerChannel	Ofrece servicios SDP al dispositivo remoto.
SDPServerChannelCreator	Esta clase implementa la interfaz *L2CAPChannelFactory* y devuelve y retorna una nueva instancia de *SDPServerChannel.*

Diagrama de Clases:

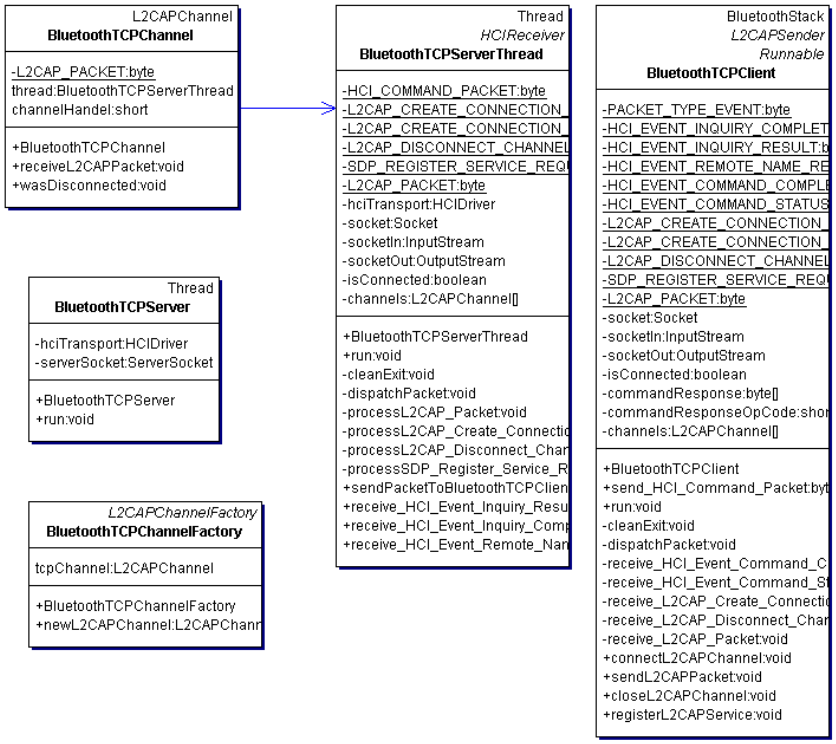


C.2.9 PAQUETE org.javabluetooth.distributed

CLASES

BluetoothTCPChannel	Esta clase es una Implementación de la interfaz *L2CAPChannel* usada por la clase *BluetoothTCPServer*.
BluetoothTCPChannelFactory	La clase *BluetoothTCPServer* la utiliza para crear objetos *BluetoothTCPChannel*.
BluetoothTCPClient	Esta aplicación del stack de protocolos de Bluetooth se conecta a través de TCP con el *BluetoothTCPServer*.
BluetoothTCPServer	Escucha las conexiones TCP entrantes sobre el puerto especificado y crea un nuevo objeto *BluetoothTCPServerThread* que maneja las conexiones una vez se han establecido.
BluetoothTCPServerThread	Gestiona una conexión TCP con un *BluetoothTCPClient.*

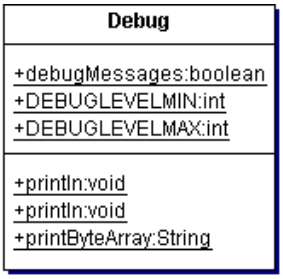
Diagrama de clases:



C.2.10 PAQUETE org.javabluetooth.util

CLASES	
Debug	Esta clase contiene métodos que permiten imprimir mensajes de depuración.

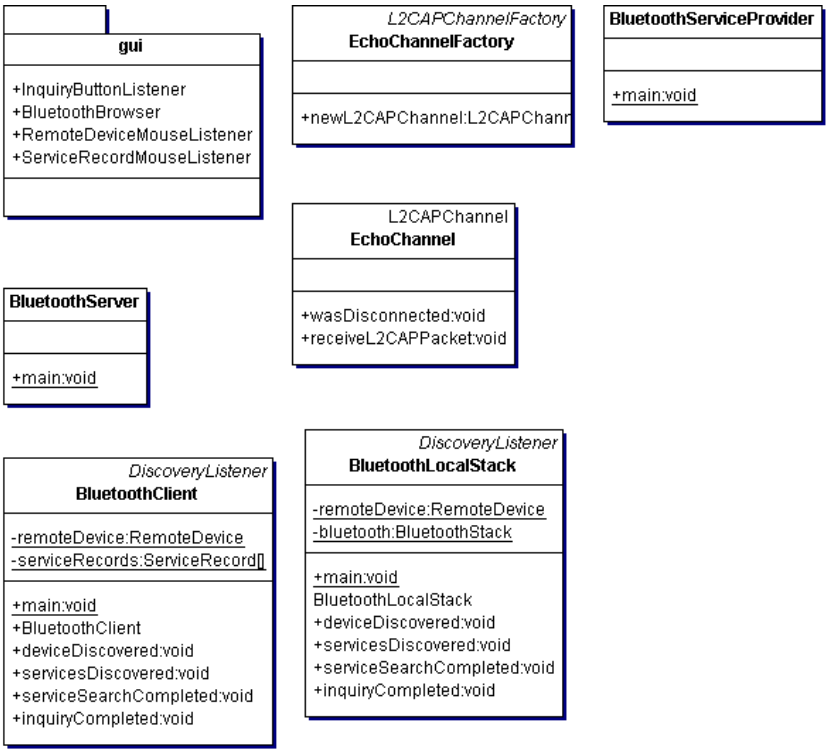
Diagrama de Clases:



C.2.11 PAQUETE org.javabluetooth.demo

CLASES	
BluetoothClient	La aplicación del cliente implementa la clase DiscoveryListener.
BluetoothLocalStack	Aplicación del stack de protocolos de Bluetooth para el dispositivo local.
BluetoothServer	Aplicación del dispositivo servidor.
BluetoothServerProvider	Demuestra el uso del BluetoothTCPClient para proporcionar servicios.
EchoChannel	Demuestra la manera de implementar los canales L2CAP.
EchoChannelFactory	Demuestra la forma de cómo escribir un ChannelFactory para ofrecer servicios Bluetooth.

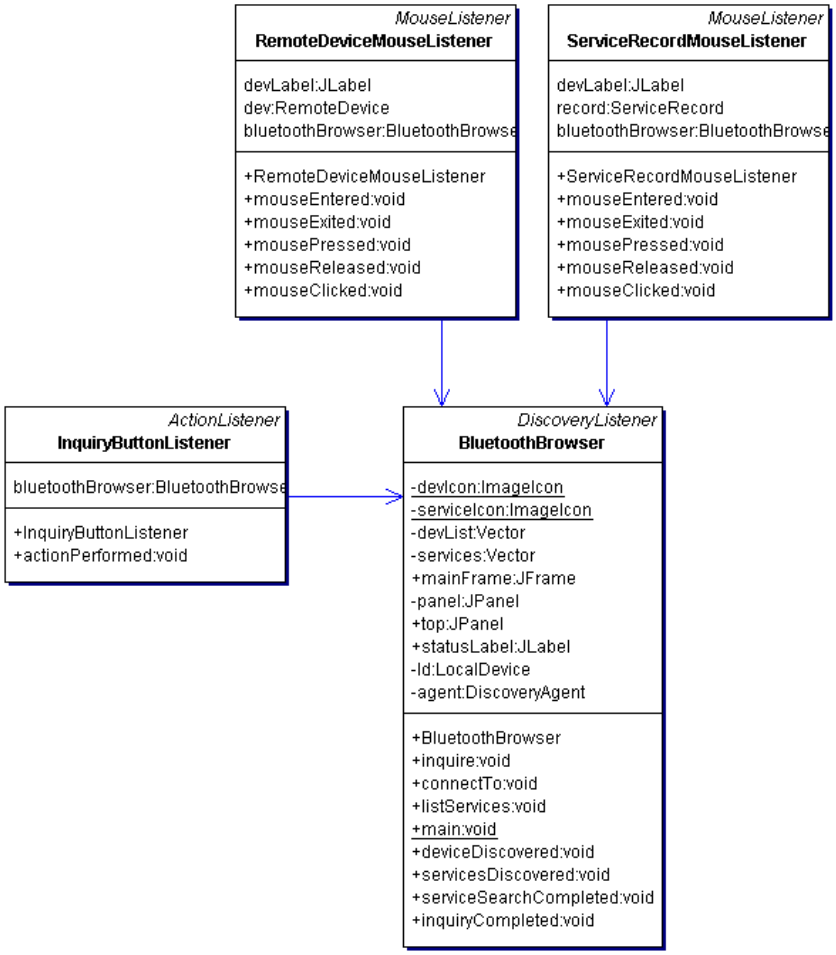
Diagrama de clases:



C.2.12 PAQUETE org.javabluetooth.demo.gui

CLASES	
BluetoothBrowser	Esta aplicación se utiliza como Interfaz Grafica de Usuario para demostrar el Stack de Protocolos de Bluetooth.
InquiryButtonListener	Corresponde al botón acción *Listener* utilizada para la indagación en el *BluetoothBrowser*.
RemoteDeviceMouseListener	Permite escuchar los cambios cuando el mouse se pone sobre el icono de un dispositivo Bluetooth.
ServiceRecordMouseListener	Permite escuchar los cambios cuando el mouse se pone sobre el icono de un servicio Bluetooth.

Diagrama de clases:


MouseListener
RemoteDeviceMouseListener
devLabel:JLabel
dev:RemoteDevice
bluetoothBrowser:BluetoothBrowse
+RemoteDeviceMouseListener
+mouseEntered:void
+mouseExited:void
+mousePressed:void
+mouseReleased:void
+mouseClicked:void
MouseListener
ServiceRecordMouseListener
devLabel:JLabel
record:ServiceRecord
bluetoothBrowser:BluetoothBrowse
+ServiceRecordMouseListener
+mouseEntered:void
+mouseExited:void
+mousePressed:void
+mouseReleased:void
+mouseClicked:void
ActionListener
InquiryButtonListener
bluetoothBrowser:BluetoothBrowse
+InquiryButtonListener
+actionPerformed:void
DiscoveryListener
BluetoothBrowser
-devIcon:ImageIcon
-serviceIcon:ImageIcon
-devList:Vector
-services:Vector
+mainFrame:JFrame
-panel:JPanel
+top:JPanel
+statusLabel:JLabel
-ld:LocalDevice
-agent:DiscoveryAgent
+BluetoothBrowser
+inquire:void
+connectTo:void
+listServices:void
+main:void
+deviceDiscovered:void
+servicesDiscovered:void
+serviceSearchCompleted:void
+inquiryCompleted:void