


Programación de Dispositivos Móviles




Sesión 18:
Conexiones avanzadas

Java y Dispositivos Móviles

© 2007-2009 Depto. Ciencia Computación e IA

Conexiones avanzadas-1

Índice




- Bluetooth
- Servicios Web

Java y Dispositivos Móviles

© 2007-2009 Depto. Ciencia Computación e IA

Conexiones avanzadas-2

Conexiones de red



- Bluetooth
- Servicios Web

Java y Dispositivos Móviles

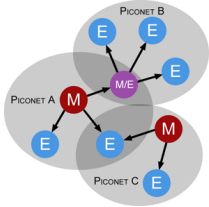
© 2007-2009 Depto. Ciencia Computación e IA

Conexiones avanzadas-3

Redes bluetooth



- Las redes bluetooth son redes “ad hoc”
 - La red se crea dinámicamente
 - Tenemos la capacidad de “descubrir” dispositivos
 - Conecta pequeños dispositivos (sustituye al cable)
- Topología



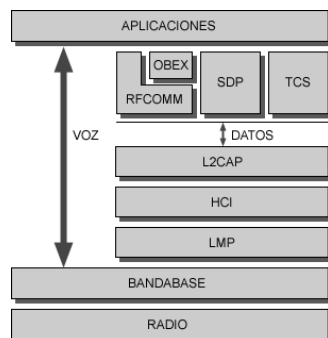
Piconet: 1 maestro con 7 esclavos como máximo

Scatternet: Conexión de varias piconets

Protocolos bluetooth



- L2CAP**
 - Protocolo a bajo nivel
 - Transmisión de paquetes
 - Sin control de flujo
- RFCOMM**
 - Puerto serie sobre radio
 - Realiza control de flujo
- SDP**
 - Descubrimiento de dispositivos



Servicios



- Los servicios se identifican mediante un UUID
 - Universal Unique Identifier
 - Clave de 128 bits, única en el tiempo y en el espacio
 - Se puede generar con herramientas como `uuidgen`
- Podemos buscar dispositivos y explorar los servicios que ofrecen
 - Los servicios se buscarán mediante su UUID
 - Tipos de búsqueda de dispositivos:

GIAC: General.

- Encuentra tanto dispositivos descubribles GIAC como LIAC.

LIAC: Limitada. Para búsquedas acotadas.

- Sólo encuentra dispositivos descubribles LIAC.

Publicar un servicio (servidor)



- Generar UUID para nuestro servicio

```
public final static String UUID =  
"00000000000010008000123456789ab";
```

- Hacemos nuestro dispositivo local descubrible

```
LocalDevice ld = LocalDevice.getLocalDevice();  
ld.setDiscoverable(DiscoveryAgent.GIAC);
```

- Crear servicio

```
StreamConnectionNotifier scn =  
(StreamConnectionNotifier)  
Connector.open("btspp://localhost:" + UUID );
```

- Aceptar conexiones entrantes

```
StreamConnection sc =  
(StreamConnection)scn.acceptAndOpen();  
InputStream is = sc.openInputStream();  
OutputStream os = sc.openOutputStream();
```

Descubrir dispositivos y servicios (cliente)



- Obtener agente de descubrimiento

```
LocalDevice ld = LocalDevice.getLocalDevice();  
DiscoveryAgent da = ld.getDiscoveryAgent();
```

- Crear un objeto `DiscoveryListener`

```
deviceDiscovered(RemoteDevice rd, DeviceClass dc);  
inquiryCompleted(int tipo);  
servicesDiscovered(int transID, ServiceRecord[] srvs);  
serviceSearchCompleted(int transID, int estado);
```

- Comenzar búsqueda de dispositivos

```
da.startInquiry(DiscoveryAgent.GIAC, mListener);
```

- Buscar servicios de un dispositivo

```
da.searchServices(null, new UUID[]{new UUID(UUID,false)},  
(RemoteDevice)obtenerDispositivoRemoto(),mListener);
```

Conectar a un servicio (cliente)



- Una vez descubiertos los servicios de nuestro entorno, habremos obtenido varios objetos `ServiceRecord` con cada uno de estos servicios

```
ServiceRecord rs =  
(ServiceRecord)obtenerServicioRemoto();
```

- Obtener URL de conexión al servicio

```
String url = rs.getConnectionURL(  
ServiceRecord.NOAUTHENTICATE_NOENCRYPT, true);
```

- Establecer la conexión

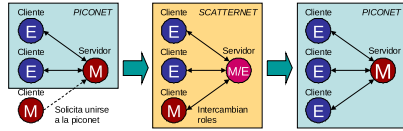
```
StreamConnection sc =  
(StreamConnection)Connector.open(url);  
InputStream is = sc.openInputStream();  
OutputStream os = sc.openOutputStream();
```

Roles maestro/esclavo



- Quien realiza la conexión (cliente) actuará como maestro
 - Indicando “;master=true” en la URL de quien publica el servicio (servidor) podemos forzar a que sea éste quien se comporte como maestro

```
Connector.open("btspp://localhost:" + UUID + ";master=true");
```

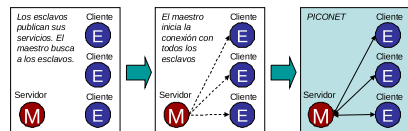


- Esto no funciona en dispositivos que no soporten el intercambio de roles maestro/esclavo

Conexión punto-a-multipunto



- Para poder hacer conexiones punto-a-multipunto en estos casos deberemos:
 - Abrir varios dispositivos que publiquen servicios.
 - En este caso, éstos serán los “clientes”.
 - Iniciar la conexión a todos estos clientes desde un único maestro
 - Este maestro será el “servidor”, ya que es quien coordinará las múltiples conexiones de los clientes.



Seguridad en bluetooth



- Podemos forzar la utilización de diferentes tipos de seguridad en las conexiones bluetooth:
 - Autenticación (;authenticate=true)
 - Los usuarios de los móviles deben conocerse
 - Se resuelve mediante “emparejamiento” (pairing)
 - Los usuarios de los móviles que se conectan deben introducir un mismo código secreto en sendos dispositivos
 - Autorización (;authorize=true)
 - Quien solicita la conexión a un servicio debe tener autorización
 - Si no está en una lista de dispositivos de confianza, se preguntará al usuario si acepta la conexión
 - Encriptación (;encrypt=true)
 - Los datos se transmiten encriptados
 - Requiere estar autenticado

Conexiones de red

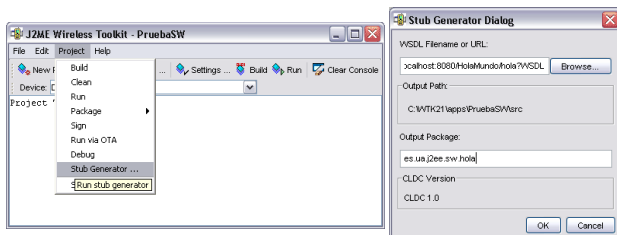


- Bluetooth
- Servicios Web

Servicios web



- Podemos acceder a servicios web desde dispositivos mediante Web Services API
 - Los servicios deben ser de tipo `document/literal`
- Podemos generar un stub mediante WTK 2.2



Invocación de servicios



- Utilizamos el stub para acceder al servicio

```
HolaMundoIF hola = new HolaMundoIF_Stub();  
try {  
    String saludo = hola.saluda("Miguel");  
} catch (RemoteException re) { // Error }
```

- Los servicios web requieren
 - Gran cantidad de memoria
 - Gran cantidad de procesamiento
 - Gran cantidad de tráfico por la red (XML)
- Esto los hace poco adecuados para los dispositivos actuales
