DISEÑO DE UN PROTOTIPO DE SERVICIO TELEMÁTICO PARA BLUETOOTH BASADO EN LA PLATAFORMA JAVA

MARY LUZ CHICANGANA FIGUEROA

JAIRO FABIAN BASTO PAREDES

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Línea de Investigación en Ingeniería de Sistemas Telemáticos
Popayán, Marzo de 2004

DISEÑO DE UN PROTOTIPO DE SERVICIO TELEMÁTICO PARA BLUETOOTH BASADO EN LA PLATAFORMA JAVA

MARY LUZ CHICANGANA FIGUEROA

JAIRO FABIAN BASTO PAREDES

Monografía para optar el título de
Ingeniero en Electrónica y Telecomunicaciones

Director: **Ing. Javier Alexander Hurtado Guaca**
Ingeniero en Electronica y Telecomunicaciones

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Línea de Investigación en Ingeniería de Sistemas Telemáticos
Popayán, Marzo de 2004

Gracias:

A Dios porque me dio la bendición de estar en este maravilloso mundo.

A mi familia, su esfuerzo esta mas arriba de todos mis logros.

A Vicky, mi amor, porque gracias a ella las cosas tienen un matiz diferente.

A mis amigos, representan fundamentalmente la realización de muchos de mis sueños.

A la vida por darme la oportunidad de VIVIR.

Fabián

A Dios por su eterna compañía, constante aliento y por el instante de tiempo que me brinda para vivir y ser feliz..

A mis padres por su confianza, amistad, apoyo y brindarme lo más puro de su amor.

A mi hermano por su alegría y paciencia.

A Richard por su compañía y apoyo incondicional.

A mis amigos por los buenos momentos compartidos y su compañía sincera.

Mary Luz

AGRADECIMIENTOS

Esta meta es el comienzo de todas las cosas buenas que están por venir y por ello solo queda tener en mente que todo esta a nuestro alcance y el hecho esta en ponerle amor a las cosas que hacemos.

Agradecemos a todas las personas que hicieron posible la culminación de este trabajo; entre ellos al Ingeniero Javier Alexander Hurtado por la motivación y la iniciativa que demostró durante el desarrollo del mismo.

Al Ingeniero Oscar Mauricio Caicedo por su inmensa colaboración y su amistad.

Al Ingeniero Carlos Enrique Serrano porque representa una luz en este camino.

Al Ingeniero Rafael Rengifo por su amistad y abierta colaboración.

A todos nuestros compañeros y amigos por su apoyo y amistad incondicional.

TABLA DE CONTENIDO

LISTA DE FIGURAS

INTRODUCCION

La concepción de un mundo completamente inalámbrico hace inminente la aparición de nuevas soluciones a necesidades relacionadas intrínsecamente con el crecimiento del mercado de dispositivos móviles que día a día hacen más fácil la vida de muchas personas en el mundo y dicho crecimiento hace parte fundamental de la sociedad. Dentro del marco de las tecnologías inalámbricas, Bluetooth se presenta como una de las alternativas para la interconexión de equipos en red, teléfonos móviles, dispositivos personales y otros equipos electrónicos que hacen parte de lo que se ha definido como una Red de Area Personal PAN (Personal Area Network), presentando una serie de ventajas que han sido de gran importancia para que entre los años 1999 y 2000 mas de 1600 empresas que hacen parte del Grupo de Interés Especial de Bluetooth (Special Interest Group - SIG), respalden dicha tecnología y contribuyan con su desarrollo.

Dadas las perspectivas frente a esta nueva tecnología y por iniciativa de algunas empresas han surgido herramientas o especificaciones de APIs[1] Bluetooth soportadas en diferentes plataformas tales como Visual C, C++ o Java las cuales permiten explorar las posibilidades que la tecnología ofrece para la creación de nuevos servicios completamente personalizados. En el momento ya se cuenta con una gran variedad de productos completamente innovadores para el sector de las telecomunicaciones que soportan Bluetooth y hoy en día se sigue trabajando para brindar un entorno de desarrollo completamente estandarizado para la implementación de nuevas aplicaciones que permitan la interoperabilidad entre todos los dispositivos pertenecientes a diferentes fabricantes.

La especificación de la tecnología Bluetooth 1.0 se divide en dos partes, el núcleo y los perfiles. El núcleo de la especificación explica cómo trabaja la tecnología. Los perfiles son modelos de aplicaciones que permiten caracterizar los servicios de forma genérica. Con el objetivo de lograr este entorno de desarrollo estándar, la especificación maneja una serie de conceptos importantes entre los que se destacan la definición de algunos *modelos de uso* [2]que permiten garantizar la interoperabilidad entre dispositivos Bluetooth fabricados por diferentes empresas que hoy en día hacen parte del SIG.

---

[1] Interfaces de Programación de Aplicaciones - Application Programming Interface

[2]Los modelos de uso son escenarios donde pueden tener lugar las aplicaciones Bluetooth.

CAPÍTULO 1 NUCLEO DE LA ESPECIFICACIÓN BLUETOOTH Y LOS PERFILES

1.1 APRECIACIÓN DEL NÚCLEO DE LA ESPECIFICACIÓN BLUETOOTH

Bluetooth se define como un estándar global de comunicación inalámbrica, que permite la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

La tecnología Bluetooth comprende hardware, software y requerimientos de interoperabilidad, por lo que para su desarrollo ha sido necesaria la participación de los principales fabricantes de los sectores de las telecomunicaciones y la informática. Posteriormente se han ido incorporando más compañías, y se espera que próximamente lo hagan también empresas de sectores tan variados como automatización industrial, maquinaria, ocio y entretenimiento, fabricantes de juguetes, electrodomésticos, etc., con lo que en poco tiempo se presentará un panorama de total conectividad entre dispositivos tanto en el hogar como en el trabajo.

En 1994 la compañía Ericsson inició un estudio para investigar la viabilidad de una interfaz vía radio, de bajo costo y bajo consumo, para la interconexión entre teléfonos móviles y otros accesorios con la intención de eliminar cables entre aparatos. El estudio partía de un largo proyecto que investigaba sobre unos multicomunicadores conectados a una red celular, hasta que se llegó a un enlace de radio de corto alcance, llamado *MC link.* Conforme éste proyecto avanzaba se fue viendo claro que éste tipo de enlace podía ser utilizado ampliamente en un gran número de aplicaciones, ya que tenia como principal virtud el que se basaba en un chip de radio relativamente económico.

A comienzos de 1997, según avanzaba el proyecto MC link, Ericsson fue despertando el interés de otros fabricantes de equipos portátiles. Para que el sistema tuviera éxito, un gran número de

dispositivos deberían estar equipados con ésta tecnología. Esto fue lo que originó a principios de 1998, la creación de un grupo de interés especial SIG, formado por 5 promotores que fueron: Ericsson, Nokia, IBM, Toshiba e Intel. La idea era lograr un conjunto adecuado de áreas de negocio, dos líderes del mercado de las telecomunicaciones, dos líderes del mercado de los PCs portátiles y un líder en la fabricación de chips. El propósito principal del consorcio fué establecer un estándar para la *interfaz* aérea junto con su software de control, con el fin de asegurar la interoperabilidad de los equipos entre los diversos fabricantes.

El primer objetivo para los productos Bluetooth de primera generación eran los entornos de la gente de negocios que viaja frecuentemente; por lo que se debería pensar en integrar el chip de radio Bluetooth en equipos como: PCs portátiles, teléfonos móviles, PDAs y auriculares. Esto originaba una serie de cuestiones previas que deberían solucionarse:

- El sistema debería operar en todo el mundo: Para poder operar en todo el mundo es necesaria una banda de frecuencia abierta a cualquier sistema de radio independientemente del lugar del planeta donde nos encontremos. Sólo la banda ISM (médico-científica internacional) de 2,45 Ghz cumple con éste requisito, con rangos que van de los 2.400 Mhz a los 2.500 Mhz, y solo con algunas restricciones en países como Francia, España y Japón. En **Colombia** el *Ministerio de Comunicaciones* reglamenta su uso así: “Las bandas, 902 - 924 MHz, 2 400 - 2 483,5 MHz y 5 725 - 5 850 MHz se atribuyen a título secundario, conforme con la resolución 3382 de 15 de diciembre de 1995, para los sistemas de espectro ensanchado”[3]

- El emisor de radio debería consumir poca energía, ya que debe integrarse en equipos alimentados por baterías: Debido a que la banda ISM es una banda abierta, el sistema de radio Bluetooth debe estar preparado para evitar las múltiples interferencias que se puedan producir. Éstas pueden ser evitadas utilizando un sistema que busque una parte no utilizada del espectro o un sistema de salto de frecuencia. En los sistemas de radio Bluetooth se suele utilizar el método de salto de frecuencia debido a que ésta tecnología puede ser integrada en equipos de baja potencia y bajo coste. Este sistema divide la banda de frecuencia en varios canales de salto, donde, los transreceptores, durante la conexión van cambiando de uno a otro canal de salto de manera pseudo-aleatoria. Con esto se consigue que el ancho de banda instantáneo sea muy pequeño y también una propagación efectiva sobre el total de ancho de banda. En conclusión, con el sistema de Salto de frecuencia (Frequency Hopping - FH), se pueden conseguir transreceptores de banda estrecha con una gran inmunidad a las

[3] MINISTERIO DE COMUNICACIONES DE COLOMBIA. CuadroAtribucion.pdf [Archivo PDF], Disponible en Internet: < URL: http://www.mincomunicaciones.gov.co >.

interferencias. *Bluetooth* utiliza 79 canales de radio frecuencia con un ancho de banda de 1MHz cada uno y una rata máxima de símbolos de 1 MSímbolo/s. Después de que cada paquete es enviado en una determinada frecuencia de transmisión, ésta cambia a otra de las 79 frecuencias. El rango típico de operación de *Bluetooth* es menor a 10 m, sin embargo se pueden alcanzar distancias de hasta 100 m con el uso de amplificadores.

Como se puede observar en la Figura 1.1, la comunicación sobre Bluetooth se divide en varias capas. A continuación se presenta una breve descripción de algunas de ellas y se tratarán en mayor detalle en capítulos posteriores.

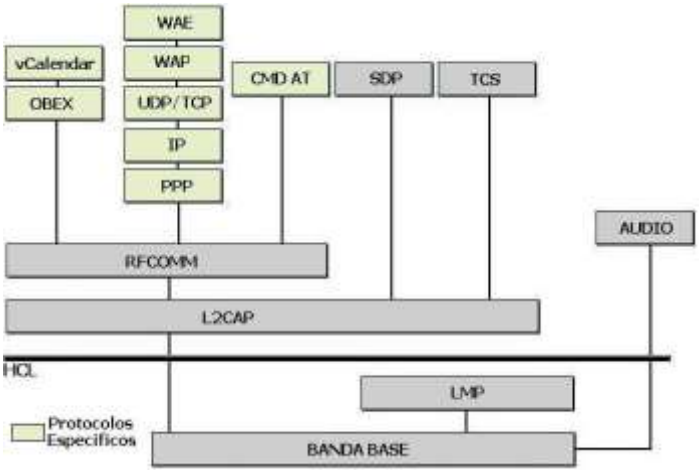


Figura 1.1 Stack de Protocolos Bluetooth

La capa de comunicación más baja es llamada **banda base**. Esta capa implementa el canal físico real. Emplea una secuencia aleatoria de saltos a través de 79 frecuencias de radio diferentes. Los paquetes son enviados sobre el canal físico en una frecuencia de salto diferente. La *Banda Base* controla la sincronización de las unidades *Bluetooth* y la secuencia de saltos en frecuencia, además es la responsable de la información para el control del enlace a bajo nivel como el reconocimiento, control de flujo y caracterización de carga útil y soporta dos tipos de enlaces: *síncrono orientado a la conexión* (Synchronous Connection Oriented - SCO), para datos y *asíncrono no orientado a la conexión* (Asynchronous Connectionless - ACL), principalmente para audio. Los dos pueden ser multiplexados para usar el mismo enlace *RF*. Usando ancho de banda reservado, los enlaces *SCO* soportan tráfico de voz en tiempo real.

El **Link Manager Protocol (LMP)** o *Protocolo de Gestión de Enlace* es el responsable de la autenticación, encriptación, control y configuración del enlace. El *LMP* también se encarga del

manejo de los modos y consumos de potencia, además soporta los procedimientos necesarios para establecer un enlace *SCO*.

El **Host Controller Interface (HCI)** o *Interfaz del Controlador de Enlace* brinda un método de interfaz uniforme para acceder a los recursos de hardware de *Bluetooth*. Éste contiene una interfaz de comando para el *controlador banda base* y la *gestión de enlace* y para acceder al hardware.

El **Logical Link Control and Adaptation Protocol (L2CAP)** o *Protocolo de Control y Adaptación de Enlace Lógico*, corresponde a la capa de enlace de datos. Ésta brinda servicios de datos orientados y no orientados a la conexión a las capas superiores. *L2CAP* multiplexa los protocolos de capas superiores con el fin de enviar varios protocolos sobre un canal banda base. Con el fin de manipular paquetes de capas superiores más grandes que el máximo tamaño del paquete banda base, *L2CAP* los segmenta en varios paquetes banda base. La capa *L2CAP* del receptor reensambla los paquetes banda base en paquetes más grandes para la capa superior. La conexión L2CAP permite el intercambio de información referente a la calidad de la conexión, además maneja grupos, de tal manera que varios dispositivos pueden comunicarse entre sí.

El protocolo **RFCOMM** ofrece emulación de puertos seriales sobre el protocolo *L2CAP. RFCOMM* emula señales de control y datos *RS-232* sobre la banda base *Bluetooth.* Éste ofrece capacidades de transporte a servicios de capas superiores (por ejemplo *OBEX*) que usan una línea serial como mecanismo de transporte. *RFCOMM* soporta dos tipos de comunicación, directa entre dispositivos actuando como **endpoints** y **dispositivo-modem-dispositivo**, además tiene un esquema para emulación de **null modem.**

El **Sevice Discovery Protocol (SDP)** o *Protocolo de Descubrimiento de Servicio* define cómo actúa una aplicación de un cliente *Bluetooth* para descubrir servicios disponibles de servidores *Bluetooth*, además de proporcionar un método para determinar las características de dichos servicios.

El **control de telefonía binario (TCS binario)** es un protocolo que define la señalización de control de llamadas, para el establecimiento y liberación de una conversación o una llamada de datos entre unidades *Bluetooth.* Además, éste ofrece funcionalidad para intercambiar información de señalización no relacionada con el progreso de llamadas.

La capa de **Audio** es una capa especial, usada sólo para enviar audio sobre *Bluetooth.* Las transmisiones de audio pueden ser ejecutadas entre una o más unidades usando muchos modelos

diferentes. Los datos de audio no pasan a través de la capa *L2CAP,* pero sí directamente después de abrir un enlace y un establecimiento directo entre dos unidades *Bluetooth.*

Protocolos Específicos

- **Control de telefonía – Comandos AT.** *Bluetooth* soporta un número de comandos *AT* para el control de telefonía a través de emulación de puerto serial (*RFCOMM*).
- **Protocolo Punto-a-Punto (PPP).** El *PPP* es un protocolo orientado a paquetes y por lo tanto debe usar su mecanismo serial para convertir un torrente de paquetes de datos en una corriente de datos seriales. Este protocolo corre sobre RFCOMM para lograr las conexiones punto-a-punto.
- **Protocolos UDP/TCP – IP.** Los estándares *UDP/TCP e IP* permiten a las unidades *Bluetooth* conectarse, por ejemplo a *Internet*, a través de otras unidades conectadas. Por lo tanto, la unidad puede actuar como un puente para *Internet*. La configuración *TCP/IP/PPP* está disponible como un transporte para *WAP*.
- **Wireless Aplication Protocol (WAP)** o *Protocolo de Aplicación Inalámbrica. WAP* es una especificación de protocolo inalámbrica que trabaja con una amplia variedad de tecnologías de red inalámbricas conectando dispositivos móviles a *Internet*. *Bluetooth* puede ser usado como portador para ofrecer el transporte de datos entre el cliente *WAP* y su servidor de *WAP* adyacente. Además, las capacidades de red de *Bluetooth* dan a un cliente *WAP* posibilidades únicas en cuanto a movilidad comparada con otros portadores *WAP.* Un ejemplo de *WAP sobre Bluetooth* sería un almacén que transmite ofertas especiales a un cliente *WAP* cuando éste entra en el rango de cobertura.
- **Protocolo OBEX.** *OBEX* es un protocolo opcional de nivel de aplicación diseñado para permitir a las unidades *Bluetooth* soportar comunicación infrarroja para intercambiar una gran variedad de datos y comandos. Éste usa un modelo *cliente-servidor* y es independiente del mecanismo de transporte y de la *API* **(Application Program Interface)** de transporte. *OBEX* usa *RFCOMM* como principal capa de transporte.

1.1.1 Concepto de Piconet

Dos o más dispositivos *Bluetooth* que comparten el mismo canal de conexión conforman una ***piconet***. Esta se establece a través de enlaces *punto-multipunto*, en donde uno de los dispositivos cumple el rol de ***maestro*** mientras los demás actúan como ***esclavos***. Una *piconet* puede tener un *máximo* de siete esclavos activos. La Figura 1.2 ilustra el esquema de un ejemplo de una *piconet*.

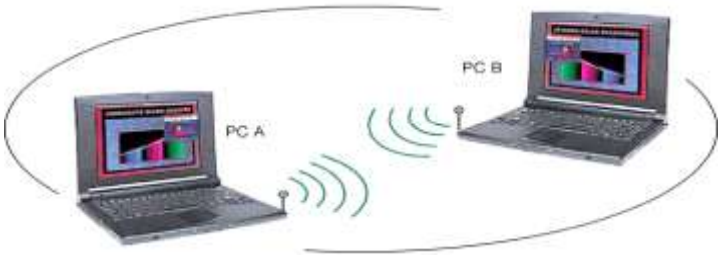


Figura 1.2 Ejemplo de una Piconet

Si un equipo se encuentra dentro del radio de cobertura de otro, éstos pueden establecer conexión entre ellos. En principio sólo son necesarias un par de unidades con las mismas características de hardware para establecer un enlace. Para regular el tráfico en el canal, una de las unidades participantes se convertirá en maestro, pero por definición, la unidad que establece la piconet asume éste papel y todos los demás serán esclavos. Los participantes pueden intercambiar los papeles si una unidad esclava quiere asumir el papel de maestro. Sin embargo, sólo puede haber un maestro en la piconet al mismo tiempo.

Cada unidad de la piconet utiliza su identidad de maestro y reloj nativo para seguir en el canal de salto. Cuando se establece la conexión, se añade un ajuste de reloj a la propia frecuencia del reloj nativo de la unidad esclava para poder sincronizarse con el reloj nativo del maestro. El reloj nativo mantiene siempre constante su frecuencia, sin embargo los ajustes producidos por las unidades esclavas para sincronizarse con el maestro, sólo son válidos mientras dura la conexión.

Los maestros controlan el tráfico en el canal, por lo cual tienen la capacidad para reservar slots en los enlaces SCO. Para enlaces ACL, se utiliza un esquema de sondeo. A un esclavo sólo se le permite enviar un slot a un maestro cuando éste se ha dirigido por su dirección MAC[4] (Control de Acceso al Medio) en el procedimiento de slot maestro-esclavo. Este tipo de slot implica un sondeo por parte del esclavo, por esta razón, en un tráfico normal de paquetes, este es enviado a una urna del esclavo automáticamente. Si la información del esclavo no está disponible, el maestro puede utilizar un paquete de sondeo para sondear al esclavo explícitamente. Los paquetes de sondeo consisten únicamente en uno de acceso y otro de cabecera. Éste esquema de sondeo central elimina las colisiones entre las transmisiones de los esclavos.

[4] La dirección MAC o dirección hardware de un dispositivo, tiene 6 bytes, es única y esta controlada por la IEEE

1.1.2 Concepto de Scatternet

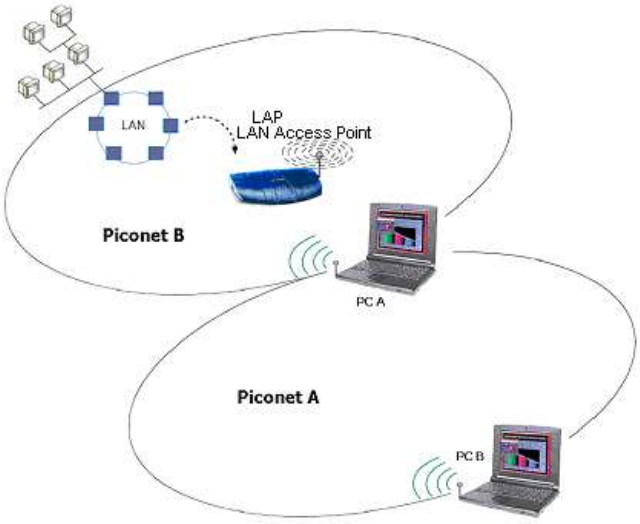


Figura 1.3 Ejemplo de una Scatternet

“Las unidades que se encuentran en el mismo radio de cobertura pueden establecer potencialmente comunicaciones entre ellas. Sin embargo, sólo aquellas unidades que realmente quieran intercambiar información comparten un mismo canal creando la piconet. Este hecho permite que se creen varias piconets en áreas de cobertura superpuestas. A un grupo de piconets se le llama scatternet. Como se puede ver en la Figura 1.3, el PC A interactúa en las dos piconets, en la Piconet A con otro PC mientras en la Piconet B se esta comunicando con un Punto de Acceso a LAN (LAN Access Point - LAP). El rendimiento, en conjunto e individualmente de los usuarios de una scatternet es mayor que el que tiene cada usuario cuando participa en un mismo canal de 1 Mhz. Además, estadísticamente se obtienen ganancias por multiplexación y rechazo de canales de salto. Debido a que individualmente cada piconet tiene un salto de frecuencia diferente, diferentes piconets pueden usar simultáneamente diferentes canales de salto.

Se debe tener en cuenta que cuantas más piconets se añaden a la scatternet, el rendimiento de la red disminuye poco a poco, existiendo una reducción por termino medio del 10%. Sin embargo el rendimiento que finalmente se obtiene de múltiples piconets supera al de una piconet puesto que hay canales de comunicación diferentes para ser aprovechados y el volumen de datos se incrementa.

1.1.3 Comportamiento de un dispositivo Bluetooth

Un dispositivo Bluetooth obedece a un comportamiento descrito por estados. Cuando dos elementos ó dispositivos coinciden dentro del alcance de sus radios, cada uno empieza a inspeccionar qué otro elemento se encuentra a su alcance, qué servicios ofrece y cuál es su dirección física. Esto ocurre dentro de un estado de inspección ó *INQUIRY* como se puede observar en la Figura 1.4.

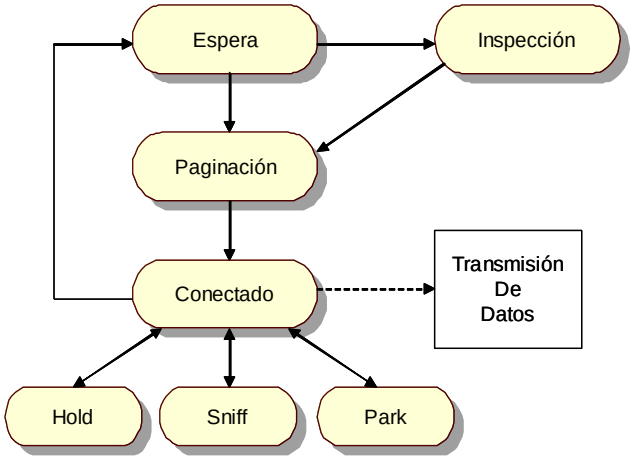


Figura 1.4 Estados del dispositivo Bluetooth

Una vez terminada la fase de inspección ya se tienen las direcciones de los dispositivos y estos pueden empezar a transmitir, no sin antes realizar una paginación ó *PAGING*, mediante la cual se establece la conexión con algún dispositivo encontrado. El estado de conexión es realmente el momento en el que se transfieren los datos y una vez terminada la transferencia, se puede retornar al estado de espera ó *STANDBY*, ó bien quedar en alguno de los estados especiales de bajo consumo de energía del dispositivo Bluetooth: SNIFF[5], HOLD[6] y PARK[7]. Aquí se presenta otro mecanismo para proporcionar bajo consumo de potencia, y por ello estos tres estados se diferencian en qué tan reactivo se vuelve el sistema ante señales externas y qué consumo de potencia requiere.

1.2 MODELOS DE USO Y PERFILES DE APLICACIÓN BLUETOOTH

El estándar *Bluetooth* fue creado para ser usado por un gran número de fabricantes e implementado en áreas ilimitadas. Para asegurar que todos los dispositivos que usen *Bluetooth*

[5] Estado en el cual el esclavo y el maestro se ponen de acuerdo para transmitir y recibir en ranuras de tiempo determinadas.

[6] En este estado el dispositivo se mantiene inactivo durante un intervalo de tiempo, sin importar que llegue o no información.

[7] En este estado el dispositivo deja de participar en la Picored y queda en un estado de actividad mínima, pero no abandona por completo la receptividad de eventos.

sean compatibles entre sí son necesarios esquemas estándar de comunicación en las principales áreas. Para evitar diferentes interpretaciones del estándar *Bluetooth* acerca de cómo un tipo específico de aplicación debería ser implementado, el *Bluetooth SIG,* ha definido modelos de uso y perfiles de protocolo.

1.2.1 Modelos de uso

Los modelos de uso son escenarios donde pueden tener lugar las aplicaciones Bluetooth, es decir, la tecnología Bluetooth además de ofrecer la posibilidad de reemplazar los cables que se usan para transmisión de datos entre distintos dispositivos digitales como computadores, impresoras, teléfonos móviles, PDAs, FAX, entre otros; brinda también la capacidad de crear redes entre personas que tienen sus propios equipos habilitados para Bluetooth, como se aprecia en la Figura 1.5.

Figura 1.5 Modelos de uso Bluetooth

1.2.1.1 Computador sin cables

Es el escenario que mejor describe la concepción de Bluetooth como una tecnología para reemplazar cables que transportan datos. Un ejemplo del gran volumen que ocupan estos cables se puede ver en un simple computador de escritorio. En este caso, los periféricos, tales como el ratón, teclado, impresoras, dispositivos de juegos, parlantes, escáners, etc. podrían tener un alto grado de libertad al no estar conectados por cables y en cambio hacer uso de enlaces de radio con el PC. Este mismo esquema puede emplearse al momento de compartir recursos hardware y un periférico este siendo usado por mas de un computador, por ejemplo, para el caso de una consola

de videojuegos ó una impresora, estos pueden ser de uso compartido entre 2 equipos sin necesidad de intercambiar cables de conexión.

1.2.1.2 Headsets

Los “Manos Libres” o Headsets son dispositivos que se usan para permitir a una persona mantener una conversación telefónica sin tener que sostener el equipo telefónico en sus manos, pero usualmente estos dispositivos también están sujetos a cables, restringiendo la movilidad del usuario; este también es un escenario que contempla Bluetooth, dado que soporta explícitamente aplicaciones de voz.

1.2.1.3 Teléfono 3 en 1

Usualmente la utilización de un aparato telefónico ocurre en varios escenarios, tales como teléfonos celulares, inalámbricos en el hogar, fijos en oficinas, etc. Un teléfono celular 3 en uno estaría equipado con Bluetooth y podría interactuar de las 3 maneras: como celular normal, como inalámbrico usando la línea telefónica a través de un punto de acceso de voz, ó como radio teléfono para comunicarse con otro igual en las proximidades.

1.2.1.4 Puente a Internet

Existen dos métodos para el uso de Bluetooth como un puente inalámbrico para establecer comunicación con LAN’s o Internet:

- *Marcación Telefónica:* Esta forma de acceso a Internet es muy similar a la que se emplea hoy en día: un arreglo convencional implica la presencia de un computador que use una línea telefónica para conectarse a un proveedor de servicios de Internet a través de un módem. Lo que Bluetooth suprime a este escenario es la presencia de cables (entre el módem y la línea telefónica). Mediante el uso de un Computador y un teléfono (sea éste móvil ó fijo) equipados con Bluetooth que soporten un perfil para “marcar a redes”, la conexión a Internet puede realizarse de forma totalmente inalámbrica.
- *Acceso directo a la red*: El acceso a Internet por medio de una LAN se realiza a través de una Gateway sin que haya necesidad de efectuar una marcación telefónica. El acceso directo a la red por medio de Bluetooth es posible usando un tipo de dispositivo llamado punto de acceso. Un Punto de acceso permite que otros dispositivos se conecten a él con el fin de proporcionar su acceso a una LAN. Se puede decir entonces que Bluetooth proporciona un *conector* inalámbrico para la conexión a redes.

1.2.1.5 Sincronización Automática

La sincronización automática es un ejemplo de la ventaja que existe, basada en la proximidad entre redes, para hacer más fácil una tarea ya existente. La sincronización es el proceso de reunir información de dos fuentes diferentes basándose en un conjunto de reglas por las cuales la información resultante es idéntica en las dos fuentes originales. Con Bluetooth es posible que dos dispositivos se sincronicen automáticamente siempre y cuando se encuentren en un rango de comunicación. Por ejemplo, una PDA guardada en el bolsillo de la chaqueta de una persona podría sincronizarse con el portátil de la misma persona mientras dicha persona camina por su oficina; esto con el fin de actualizar, por ejemplo, un conjunto de archivos en el portátil que se modificaron en su PDA mientras se encontraba fuera de su oficina, de este modo, la información será la misma y estará actualizada en los dos dispositivos.

1.2.1.6 Conferencia Interactiva (Transferencia de Archivos)

La transferencia de archivos está ligada generalmente al uso de cables y unidades de almacenamiento. Un entorno inalámbrico permite establecer enlaces temporales entre dispositivos de manera que puedan intercambiarse archivos y otros arreglos de datos. Es posible establecer una conferencia interactiva en donde los participantes de la reunión pueden intercambiar tarjetas de negocios ó archivos empleando portátiles que incorporan módulos Bluetooth.

1.2.2 Perfiles Bluetooth

Los perfiles de Bluetooth se han desarrollado para describir cómo se ejecutan las aplicaciones de los modelos de uso, estos perfiles son definidos como parte de la especificación Bluetooth y tienen como fin principal hacer que las aplicaciones puedan enmarcarse en tales perfiles y de esta manera disminuir el riesgo de problemas de interoperabilidad entre productos/aplicaciones de fabricantes diferentes.

1.2.2.1 Estructura de los perfiles Bluetooth

Un perfil es dependiente de otro perfil si reutiliza partes de ese perfil, referenciandolas implícitamente o explícitamente. La dependencia se ilustra en la Figura 1.6: un perfil tiene dependencia sobre el perfil o perfiles en los que se encuentra contenido directamente e indirectamente. Por ejemplo, el perfil de Acceso a una LAN es dependiente de los perfiles Puerto serial y Acceso Genérico.

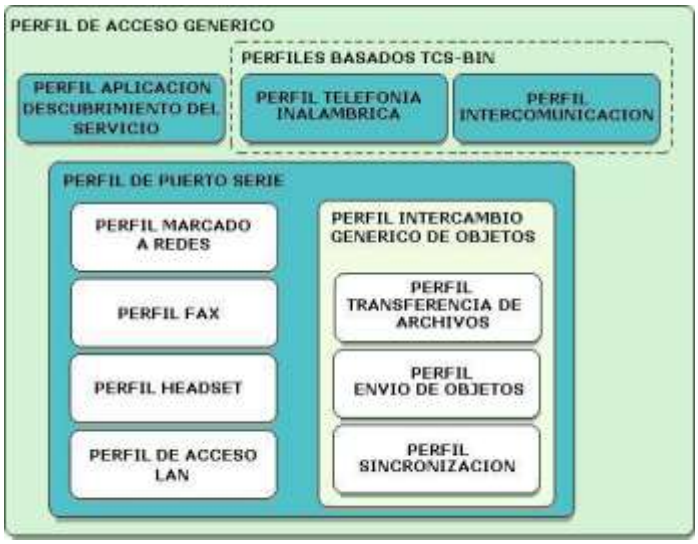


Figura 1.6 Estructura de los perfiles Bluetooth

1.2.2.2 Generic Access Profile – GAP

El Perfil de Acceso Genérico describe los procedimientos generales para el descubrimiento de dispositivos y gestión de aspectos relacionados con la conectividad entre dispositivos *Bluetooth*. Es esencialmente el perfil en el cual todos los otros Perfiles se basan.

1.2.2.3 Service Discovery Application Profile – SDAP

El Perfil de Aplicación de Descubrimiento del Servicio define las características y procedimientos para que una aplicación en un dispositivo Bluetooth pueda descubrir los servicios registrados en otros dispositivos Bluetooth y recuperar cualquier información disponible y deseada pertinente a estos servicios.

1.2.2.4 Cordless Telephony Profile – CTP

El Perfil de Telefonía Inalámbrica define cómo un teléfono móvil puede ser usado para acceder a un servicio de telefonía de red fija a través de una estación base. También se usa para telefonía inalámbrica de hogares u oficinas pequeñas. El perfil incluye llamadas a través de una estación base, haciendo que dichas llamadas se hagan directamente entre dos terminales y accediendo adicionalmente a redes externas. Es usado por dispositivos que implementan el modelo de uso "teléfono 3 en 1".

1.2.2.5 Intercom Profile – IP

El Perfil de Intercomunicación define los requerimientos necesarios para que los dispositivos Bluetooth soporten la funcionalidad del intercomunicador dentro del modelo de uso del teléfono 3 en 1. Este perfil también se refiere a cómo el 'walkie-talkie' usa Bluetooth.

1.2.2.6 Serial Port Profile – SPP

El Perfil del Puerto Serial define los requerimientos necesarios para que dispositivos *Bluetooth* puedan establecer conexiones de cable serial emuladas usando RFCOMM entre dos dispositivos similares. RFCOMM es usado para transportar los datos del usuario, señales de control de *modem* y comandos de configuración. *El perfil de puerto serial* es dependiente del *GAP.*

1.2.2.7 Headset Profile – HS

El Perfil de Manos Libres define los requerimientos necesarios para que los dispositivos *Bluetooth,* soporten el uso de manos libres. En este caso el dispositivo puede ser usado como unidad de audio inalámbrica de entrada/salida. El perfil soporta comunicación segura y no segura.

1.2.2.8 Dial-up Networking Profile – DNP

El perfil Dial-up Networking (Conexión por Línea Conmutada) define los requerimientos que usarán los dispositivos (módems, teléfonos celulares) al implementar el modelo de uso llamado “Puente a Internet”.

1.2.2.9 Fax Profile – FP

El Perfil de Fax define los requisitos necesarios para que los dispositivos Bluetooth puedan soportar el uso de Fax. Esto le permite a los teléfonos celulares Bluetooth (o módem) ser usados por un computador como un módem de fax inalámbrico para enviar/recibir un mensaje de fax.

1.2.2.10 LAN Access Profile – LAP

El perfil de acceso a una LAN define cómo Bluetooth habilita a los dispositivos para que puedan acceder a los servicios de una LAN usando el protocolo punto-punto – PPP sobre RFCOMM. *PPP* es ampliamente usado para lograr acceder a redes soportando varios protocolos de red. El perfil soporta acceso *LAN* para un dispositivo *Bluetooth* sencillo, acceso LAN para varios dispositivos Bluetooth y PC-a-PC (usando interconexión *PPP* con emulación de cable serial).

1.2.2.11 Generic Object Exchange Profile – GOEP

El Perfil Intercambio Genérico de Objetos establece la base necesaria (define los protocolos y procedimientos) para que los dispositivos Bluetooth soporten el modelo de uso de intercambio de objetos. El modelo de uso puede ser Sincronización, Transferencia de archivos, o modelo Object Push.

1.2.2.12 Object Push Profile – OPP

El Perfil Envío de Objetos es una instancia del perfil de intercambio de objetos, puesto que si el perfil de intercambio de objetos no es utilizado directamente por las aplicaciones, el perfil de envío de objetos sí es prácticamente un ejemplo de aplicación usando el protocolo OBEX.

1.2.2.13 File Transfer Profile – FTP

El Perfil de Transferencia de Archivo define los requisitos para aplicaciones que proporcionan el modelo de uso transferencia de archivos. En el modelo de uso “transferencia de archivos” existen procedimientos para chequear, transferir y manipular un grupo de objetos de otro dispositivo

Bluetooth. Los objetos bien pueden ser archivos o directorios de un grupo de objetos tal como un sistema de archivos.

1.2.2.14 Synchronization Profile – SP

El Perfil de Sincronización define los requerimientos para aplicaciones que proporcionan el modelo de uso "sincronización". Algunos escenarios típicos cubiertos por este perfil involucran sincronización manual o automática de datos cuando dos dispositivos Bluetooth están dentro del rango.

1.3 PLATAFORMA DE DESARROLLO PARA BLUETOOTH

En los últimos años Bluetooth se convirtió en un estándar de facto global para la conectividad inalámbrica, en este momento hace parte de múltiples escenarios de aplicación y esta siendo embebido en diferentes tipos de dispositivos, pero probablemente en el futuro llegará a ser un estándar para millones de teléfonos móviles, PCs, PDAs, computadores portátiles y un gran rango de otros dispositivos electrónicos. Debido a esto, algunas empresas han creado diferentes herramientas, kits y especificaciones de APIs Bluetooth soportadas en diferentes lenguajes como Visual C, C++, Java o Delphi para facilitar el desarrollo de aplicaciones innovadoras, servicios de valor agregado y soluciones punto a punto para un mercado cada vez mas exigente.

Para crear aplicaciones que tomen ventajas del número creciente de dispositivos Bluetooth en el mercado, es necesario enfocarse en tres aspectos: la portabilidad de las aplicaciones, funcionamiento Over The Air (OTA) y facilidad en la programación:

- Para que los dispositivos Bluetooth sean útiles en redes colaborativas, es esencial que las aplicaciones corran sobre muchas plataformas. Actualmente, cada pila de protocolos Bluetooth adjunta su propia API en uno o más Sistemas Operativos (OS), entre los que se encuentran Windows CE .NET, Symbian, Palmos, incluso Windows 2000 o Windows 98. Si un desarrollador implementa una aplicación para un cierto OS, es posible que dicha aplicación no tenga el mismo desempeño o funcionalidad en otro OS, convirtiéndose esto en un obstáculo en el desarrollo de aplicaciones independientes para dispositivos Bluetooth. Para lograr este nivel de portabilidad es necesario tener una API estándar Bluetooth y un entorno software que se ejecute en cualquier OS.
- Un segundo atributo importante es la funcionalidad Over The Air que se refiere a la habilidad que tiene un dispositivo móvil de descargar una o más aplicaciones a través del enlace inalámbrico que ha establecido y que dicha aplicación pueda ser ejecutada correctamente sobre el dispositivo. Por ejemplo la aplicación podría ser proporcionada a través de un enlace Bluetooth donde el usuario desde un punto de acceso Bluetooth la descargaría, haría uso de

ella y finalmente podría borrarla si así lo desea. Guías virtuales de turismo, mapas interactivos de espacios públicos y asistentes de conferencias, son algunas de las posibles aplicaciones.

- Para que durante el desarrollo de aplicaciones punto-a-punto se considere a Bluetooth como una buena opción, resulta conveniente que el estándar Bluetooth sea de fácil uso. Esto significa que la tecnología permita llevar a cabo el proceso de desarrollo a través de diferentes módulos independientes pero que a su vez deben ser interoperables a la hora de implementar algún servicio, y que exista algún leguaje o plataforma que soporte los requerimientos dados.

Aunque existen muchas opciones a la hora de elegir la plataforma de desarrollo para elaborar nuevas y mejores aplicaciones Bluetooth, la plataforma que resulta mas adecuada para cumplir con los aspectos mencionados anteriormente es JAVA puesto que además de ofrecer ciertas características importantes, abre la posibilidad a una nueva generación de aplicaciones Bluetooth. Algunas de las características que ofrece Java son las siguientes:

- Las aplicaciones Java son interpretadas por la Maquina Virtual de Java (VM) y se representan en un formato bytecode estándar. Por lo tanto, cualquier dispositivo que tenga una VM puede interpretar y ejecutar bytecode's de Java. Entonces, las aplicaciones pueden compartirse a través de enlaces Bluetooth, así como si la aplicación enviara datos usando estos enlaces.
- Con el desarrollo de la tecnología, se han construido muchos módulos que le permiten a los desarrolladores y diseñadores de sistemas distribuidos complejos concentrarse en el problema específico de la aplicación, en lugar de tener que enfocarse en los niveles más bajos de conectividad. Java proporciona una estructura que puede usarse para este propósito - el marco de Conexión Genérico en la Plataforma Java 2 Micro Edición es un conjunto de abstracciones de fácil uso para programación de red.

Las ventajas que ofrece Java con respecto a otras herramientas son múltiples, es por esto que en marzo de 2002, fue aprobada la primera API estándar para Bluetooth - API Java para Tecnología Inalámbrica Bluetooth (Java APIs for Bluetooth Wireless Technology-JABWT) versión 1.0a. Como requerimiento de estandarización resultó la recomendación JSR[8]-82. La especificación se construye sobre el componente de la Plataforma Java 2 Micro Edición (J2ME). Esta ofrece un conjunto de abstracciones de Java para cada uno de los componentes importantes de la pila de protocolos Bluetooth - L2CAP, RFCOMM, SDP, OBEX y funcionalidad HCI, además soporta los perfiles Bluetooth fundamentales GAP, SPP, GOEP y SDAP. En los próximos meses saldrán al mercado productos Java/Bluetooth.

[8] JSR – Java Specification Request

Como ilustración de las capacidades de Java como una herramienta software para aplicaciones de Bluetooth, considere a un usuario que quiere imprimir la lista de contactos de su teléfono móvil. El usuario se acerca a una Impresora habilitada con Bluetooth. El teléfono y la impresora no se han "pre - configurado" para trabajar entre sí - el software del teléfono no incluye a un dispositivo controlador para la impresora. Sin embargo, usando Java y Bluetooth, el teléfono puede descubrir la impresora, descubrir el servicio de impresión y entonces puede pedir el controlador de Java para la impresora. El controlador se proporciona al teléfono bajo la funcionalidad Over-The-Air, que ahora tiene la posibilidad de imprimir en este tipo de dispositivos. Usando la movilidad del código y la portabilidad de Java, dos dispositivos pueden colaborar para lograr alguna tarea útil.
En capítulos posteriores se ofrece además de la recomendación JSR-82 una explicación detallada de la API.

CAPÍTULO 2 PERFILES PARA EL DESARROLLO DE UNA APLICACIÓN TELEMATICA

Los Perfiles describen cómo se usa la tecnología, es decir, cómo diferentes partes de la especificación pueden emplearse para cumplir una función deseada por un dispositivo Bluetooth. Un perfil puede describirse como una porción vertical de la pila de protocolos, que además de definir las opciones que son obligatorias para el perfil en cada protocolo, también define los rangos de los parámetros para cada protocolo. El concepto perfil se usa para disminuir el riesgo de problemas de interoperabilidad entre los productos de diferentes fabricantes. Estos perfiles normalmente no definen algo adicional a lo recomendado en la especificación Bluetooth.

Un perfil llega a ser dependiente de otro perfil si reutiliza partes de dicho perfil, referenciándolas implícitamente o explícitamente. Un perfil tiene dependencia sobre el perfil o perfiles en los que se encuentra contenido directamente e indirectamente. Como ya se había mencionado el perfil de Acceso a una LAN es dependiente de los perfiles Puerto serial y Acceso Genérico.

En la actualidad el SIG Bluetooth se encuentra en la fase final de evaluación y aprobación para lanzar 12 nuevos perfiles[9], Los nuevos perfiles están desarrollándose, aun permanecen bajo revisión y desarrollo, además no están abiertos al público en general.

Tanto los nuevos perfiles, como los perfiles existentes, pueden dividirse en diferentes categorías. En la Figura 2.1 se muestra la estructura tanto de los nuevos como los perfiles ya existentes.

[9] Para obtener mas información acerca de los nuevos perfiles visitar http://www.bluetooth.org/specifications.htm

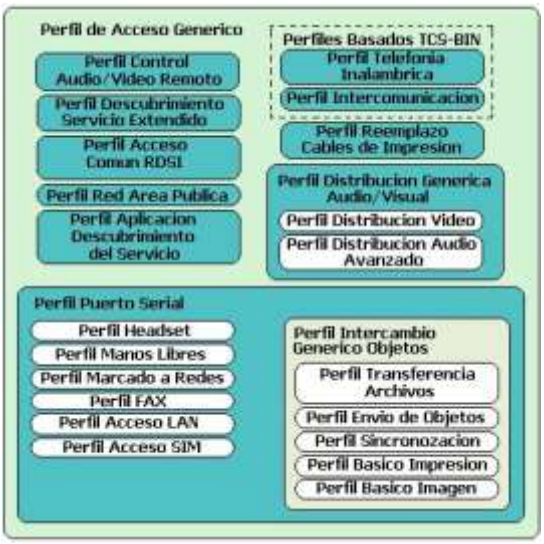


Figura 2.1 Estructura de los perfiles Bluetooth versión 2.0

Dentro de este capitulo no se va a profundizar en la descripción de estos nuevos perfiles y por el contrario, se van a explicar detalladamente los perfiles involucrados en la creación del servicio de acceso a una LAN, que se convertirá en el ejemplo para la creación de un prototipo de servicio telemático.

2.1 PERFIL DE ACCESO GENÉRICO

Un aspecto relevante al momento de conectar dos dispositivos a través de un cable de datos, es de una u otra forma la cercanía física entre ellos. Normalmente se sabe por ejemplo, que un módem externo requiere de un cable para ser conectado a un PC y también de una configuración posterior para hacer que todo funcione; pero en el caso de conexiones inalámbricas donde no podemos ver el cable, y por supuesto, los dispositivos tampoco perciben estar “conectados” mediante este cable virtual, la situación debe solucionarse suministrando a los dispositivos cierta información básica para poder entablar una comunicación. Partiendo de dicha información, un dispositivo puede estar programado para no recibir conexiones (bien sea porque se encuentre ocupado) ó también para recibirlas siempre, en este caso los dispositivos que se le conecten ó que pretendan conectársele pueden hacerlo incluso si se encuentran en recintos diferentes ó incluso bajo llave, lo cual es una posibilidad no muy agradable para dispositivos que guarden información personal ó que presten servicios, ya que cualquier otro dispositivo podría en principio atentar contra la seguridad de dicha información.

Lo anterior plantea tres problemas básicos:

- Los dispositivos deben conocer con qué otros pueden conectarse.
- Debe existir una forma de poder restringir tales conexiones.
- Es necesario proteger la integridad de la información que se almacena e intercambia.

Frente a esto, pueden plantearse soluciones a nivel de las capas más bajas de protocolos que conforman la pila de Bluetooth, y es precisamente este hecho lo que hace que estas capas tengan gran importancia al ser utilizadas de manera similar por todas las aplicaciones para ejecutar estas funciones básicas, y éste perfil esta orientado a resolver tales necesidades.

En este perfil se definen los procedimientos genéricos relacionados con el descubrimiento de dispositivos Bluetooth así como los aspectos de gestión del enlace de conexión a dispositivos Bluetooth. Además se definen los procedimientos relacionados al uso de diferentes niveles de seguridad. Esencialmente este perfil describe cómo se usan las capas mas bajas (LMP y Banda base) junto con algunas capas superiores.

2.1.1 Alcance del perfil

El propósito principal de este perfil es describir el uso de las capas mas bajas de la pila de protocolos Bluetooth (HCI y LMP). Para describir las alternativas de seguridad, también se incluyen las capas más altas (L2CAP, RFCOMM y OBEX).

2.1.2 Configuración de los Roles

Para describir los roles, que pueden tomar los dos dispositivos involucrados en una comunicación Bluetooth, la notación genérica define como **parte A** al dispositivo que inicia el proceso de paging o paginación durante el establecimiento del enlace y **parte-B** al dispositivo paginado o aceptador. La parte A es la que para un procedimiento dado inicia el establecimiento del enlace físico o inicia una transacción en un enlace existente.
El iniciador y el aceptador generalmente ejecutan los procedimientos genéricos según se especifica en este perfil.

2.1.3 Requerimientos del Usuario

El usuario Bluetooth, en principio, debe tener la capacidad de conectar un dispositivo Bluetooth a cualquier otro dispositivo Bluetooth. Aún si los dos dispositivos conectados no comparten ninguna aplicación en común, esto debe ser posible usando las capacidades básicas de Bluetooth.

2.1.4 Fundamentos del perfil

- Se debe definir los modos de operación genéricos para todos los perfiles.
- Definir los procedimientos generales que se usan para descubrir la identidad, nombre y capacidades básicas del otro dispositivo Bluetooth, el cual debe estar en un modo que pueda ser descubrible.
- Definir el procedimiento general de cómo se crean los vínculos entre dispositivos Bluetooth.
- Describir los procedimientos generales que se pueden usar para establecer conexiones con otros dispositivos Bluetooth.

2.1.5 Representación de Parámetros Bluetooth

- Dirección del dispositivo Bluetooth (BD_ADDR):

La BD_ADDR es una dirección única para cada dispositivo Bluetooth, la cual se recibe desde un dispositivo remoto durante el procedimiento de descubrimiento del dispositivo. A nivel de la Interfaz de Usuario - UI ésta es llamada “Bluetooth Device Address (Dirección del Dispositivo Bluetooth)” y esta definida en el nivel de Banda Base como una dirección de 48 bits (12 caracteres hexadecimales).

- Nombre del Dispositivo Bluetooth (Nombre Amigable al Usuario):

El nombre del dispositivo Bluetooth es una cadena que se recibe como respuesta a una solicitud hecha a nivel del Protocolo de Gestión del Enlace - LMP. A nivel de la Interfaz de Usuario - UI éste es llamado “Bluetooth Device Name (Nombre del dispositivo Bluetooth)” y puede ser hasta de 248 caracteres.

- Password Bluetooth (Número PIN):

El PIN Bluetooth es usado para autenticar dos dispositivos Bluetooth (que no han intercambiado previamente las claves de enlace) y crear una relación confiable entre ellos. El PIN puede ser ingresado a nivel de la UI pero también puede almacenarse en el dispositivo. A nivel de la Interfaz de Usuario - UI éste es llamado “Bluetooth Passkey (Password Bluetooth)’”

- Clase de Dispositivo Bluetooth (Tipo del Dispositivo):

La clase del dispositivo es un parámetro que se recibe durante el procedimiento de descubrimiento del dispositivo, además indica el tipo de dispositivo y qué tipos de servicios soporta. La información dentro del parámetro Clase de Dispositivo se refiere a “La clase de dispositivo Bluetooth” y al “Tipo de Servicio Bluetooth”.

2.1.6 Ejemplos de Uso del Acceso Genérico

Asumiendo el caso de un punto de acceso a la red que puede estar ubicado en una oficina. Un usuario de un computador portátil desea acceder a la red, pero previamente para tal fin utilizó su celular equipado con Bluetooth; esta vez el usuario tendrá la oportunidad de elegir cuál de los dos dispositivos utilizar, si el celular (lo cual representa un cargo a pagar por utilizar la red de acceso celular) ó emplear el punto de acceso, el cual puede estar conectado a una red fija (tal vez corporativa). Dado que el punto de acceso solamente debe ser descubierto por dispositivos con capacidad de procesamiento para acceder a la red, entonces para tal aplicación especifica (que tal vez requiera de un software ejecutándose como servidor en el punto de acceso) es necesario restringir la facilidad para descubrir al punto de acceso cuando se está en el mismo entorno con otros dispositivos Bluetooth.

Esto lleva a tres estados de descubrimiento: el de No Descubrimiento, el de Descubrimiento Limitado y el de Descubrimiento General. Más adelante se realizará una descripción detallada de estos estados.

Teniendo ahora el caso de una impresora ubicada en un entorno de oficina y que está equipada con un puerto inalámbrico Bluetooth, lo cual permite que varios equipos puedan acceder a sus servicios; pueden tener lugar varios escenarios de uso:

- Un computador que está fijo en la oficina accede a ella para hacer una impresión de un documento.
- Un dispositivo portátil foráneo desea hacer una impresión en el mismo aparato.
- Un equipo de una oficina contigua, gracias a que logra descubrir la impresora, desea realizar una impresión en ella.

De acuerdo a los escenarios de uso que pueden tener lugar; existirían varios tipos de dispositivos con la intención de acceder al servicio que ofrece la impresora, es por esta razón que se generan los conceptos de los distintos modos de establecimiento de enlaces y los modos de seguridad para filtrar qué dispositivos se quiere que tenga acceso al servicio. Ver con mayor detalle en las secciones 2.1.9 y 2.1.10.

2.1.7 Descubrimiento Inicial de Dispositivos y Servicios

Para que un dispositivo descubra la presencia de otro es necesario que el dispositivo remoto esté habilitado para responder a las búsquedas que efectúe el primero; esto define niveles de descubrimiento en los que puede quedar un dispositivo. Luego del descubrimiento, en el caso del

punto de acceso, el usuario necesita saber a qué dispositivo recurrir para usar el servicio de acceso a la red; por ello al ser descubierto, cada dispositivo debe ofrecer una manera de saber qué servicios ofrece y cómo conectarse a ellos; este último dato se obtiene del uso del perfil de descubrimiento, pero los servicios básicos se encuentran registrados previamente en una variable de 24 bits llamada CoD[10]. Sus bits pueden indicar si el dispositivo es un PC, un dispositivo de captura (escáner, micrófono, cámara), un dispositivo de salida (impresora, parlantes), si ofrece servicios de objetos, si es un dispositivo de localización, etc.

2.1.7.1 Modo de descubrimiento general

Operando en este modo el dispositivo está periódicamente esperando por indagaciones que utilizan como parámetro el Código de Acceso Genérico a Indagación (General Inquiry Access Code – GIAC); la utilización de este código garantiza la respuesta a todas las indagaciones que reciba su procesador de banda base, de esta manera todos los dispositivos que realicen indagaciones lo encontrarán. El modo de descubrimiento general se usa en dispositivos que requieren ser descubiertos continuamente y que no tienen condiciones específicas para ser encontrados. El propósito es responder a un dispositivo que hace una consulta general.

2.1.7.2 Modo de descubrimiento limitado

El modo de descubrimiento limitado debe usarse por dispositivos que necesitan ser descubiertos sólo por un período limitado de tiempo, durante ciertas condiciones temporales o por un evento específico. El propósito es responder a dispositivos que periódicamente hacen indagaciones, pero utilizando el código de Acceso a Indagación Limitado, el cual es un código reservado para este propósito. Un dispositivo en modo de descubrimiento limitado puede ser descubierto en menos tiempo que por dispositivos que hagan indagaciones con el código genérico. Para una aplicación específica puede ser deseable que el dispositivo en el que reside entre en este modo, puesto que así solo los dispositivos que se quiere que se conecten a tal servicio podrán descubrirlo gracias a la utilización de indagaciones con el código Limitado. Un ejemplo de ello es en servicios de objetos (transferencia de archivos, de sincronización, etc.) en los cuales solo se va a emplear un cliente específico (PDA por ejemplo) para acceder a tal servidor (Computador de escritorio); si se programa la PDA y el Computador para que utilicen el código de acceso limitado, entonces otros dispositivos (probablemente no deseados) no podrán descubrir el mismo computador y por tanto, el mismo servicio.

[10] Clase de Dispositivo

2.1.7.3 Modo de No descubrimiento

En este modo, un dispositivo puede no responder a indagaciones de otros dispositivos; puede utilizarse por diferentes razones, como por ejemplo, una carga de procesamiento alta en el dispositivo puede hacer que entre en este modo mientras se restablece su funcionamiento.

2.1.8 Procedimientos para el descubrimiento de dispositivos Bluetooth

Los procedimientos de indagación (inquiry) y descubrimiento son aplicables sólo al dispositivo que los inicia.

2.1.8.1 Indagación general

- El propósito del procedimiento de indagación general, es proporcionarle a la parte A la dirección del dispositivo Bluetooth, el reloj y la clase de dispositivo. Los dispositivos en el modo de descubrimiento general serán descubiertos usando una indagación general.
- La indagación general la deben usar dispositivos que necesiten descubrir otros dispositivos que continuamente son descubiertos.
- Para recibir respuesta de una indagación general, los dispositivos remotos que estén dentro del rango con respecto a la parte A, tienen que estar en un modo de descubrimiento ya sea limitado o general.

2.1.8.2 Indagación limitada

- El propósito del procedimiento de indagación limitada es proporcionarle a la parte A la dirección del dispositivo Bluetooth, el reloj, y la clase de dispositivo. Los dispositivos deben estar dentro del rango con respecto a la parte A, y pueden configurarse para examinar los mensajes de indagación con el Código de Acceso a Indagación Limitada, además de examinar los mensajes de indagación con el Código de Acceso a Indagación General.
- La indagación limitada debe usarse por dispositivos que necesitan descubrir otros dispositivos que pueden ser descubiertos sólo por un período de tiempo limitado, durante condiciones temporales o por un evento específico.
- Para recibir respuesta de una indagación limitada, el dispositivo remoto que este dentro del rango con respecto a la parte A debe estar en el modo de descubrimiento limitado.

2.1.8.3 Descubrimiento del nombre

El propósito del descubrimiento del nombre es ofrecerle al dispositivo iniciador el Nombre de los dispositivos Bluetooth conectables (es decir los dispositivos que están dentro del rango y que responderán a la paginación). Esto puede hacerse de 2 maneras:

- *Solicitud del Nombre*

La Solicitud del Nombre es el procedimiento para recuperar el Nombre del Dispositivo Bluetooth desde un dispositivo Bluetooth conectable. No es necesario realizar el procedimiento de establecimiento del enlace completo para obtener el nombre del otro dispositivo.

- *Descubrimiento del Nombre*

El descubrimiento del nombre es el procedimiento para recuperar el Nombre del Dispositivo Bluetooth desde dispositivos Bluetooth conectables realizando solicitudes de nombre hacia dispositivos conocidos (es decir dispositivos Bluetooth para los cuales las Direcciones de los Dispositivos Bluetooth están disponibles).

2.1.8.4 Descubrimiento del dispositivo

- El propósito del descubrimiento del dispositivo es proporcionarle a la parte A la Dirección, el Reloj, La Clase de Dispositivo y el nombre de los dispositivos Bluetooth que pueden ser descubiertos.
- Los dispositivos encontrados durante el proceso de descubrimiento de dispositivos deben estar en algún modo de descubrimiento y ser conectables.

2.1.8.5 Vinculación

- El propósito de la vinculación es crear una relación entre dos dispositivos Bluetooth basada en una clave de enlace común. La Clave del enlace es creada e intercambiada durante el proceso de vinculación y se espera que sea guardada por ambos dispositivos Bluetooth, para ser usada durante una autenticación futura.

2.1.9 Establecimiento de Enlaces

Una vez que se ha descubierto un dispositivo, se puede establecer un enlace físico con él; la respuesta a este evento también puede estar restringida; puede tomarse de nuevo el ejemplo de la impresora y los otros equipos de oficina.

2.1.9.1 Relación de Confianza Entre Dispositivos

En el establecimiento de los enlaces interviene una condición de autenticación, dependiendo de la configuración de los elementos involucrados en la comunicación. Un elemento puede solicitar autenticación a otro que quiera conectarse con él; en tal procedimiento se intercambian los números PIN y de esta manera se establece una relación de confianza entre los dispositivos,

puesto que ambos pueden conservar tales números y utilizarlos de nuevo en conexiones posteriores.

En el caso de la impresora existen computadores que durante su configuración ó durante la primera conexión que realizaron con la impresora fueron autenticados, de tal manera que impresiones posteriores puedan hacerse sin necesidad de solicitar al usuario un numero PIN de nuevo.

2.1.9.2 Modos Conectable y No Conectable

En estos modos el dispositivo puede ó no aceptar conexiones a nivel de *Link Manager*, aunque dado el caso que no acepte la conexión, sí puede responder a indagaciones. Puede ser útil en el caso de dispositivos que reciben datos de sensores inalámbricos Bluetooth, donde los sensores son los que deben aceptar las conexiones y no el dispositivo central. Cuando un dispositivo Bluetooth esta en el modo no conectable, no responde al procedimiento de paginación; en el caso de estar en el modo conectable, si responde al proceso de paginación.

2.1.9.3 Modos Autenticable y No Autenticable

En estos modos se emplea la capacidad de autenticar otros dispositivos como característica principal. Cuando un dispositivo solicita conectarse a otro, el segundo puede solicitar autenticación para dar paso a la transacción; el primer dispositivo, dependiendo del modo en que se encuentre puede aceptar ó rechazar la petición de autenticación; un dispositivo puede estar programado en modo *No Autenticable* si no tiene las capacidades de despliegue ó de teclado para visualizar y entrar el número PIN.

2.1.10 Aspectos de seguridad

El inconveniente de proteger la integridad tanto de la información que se almacena como la que se transporta y también de evitar la suplantación por parte de dispositivos no deseados para acceder a algunos servicios de carácter privados, hace necesario establecer modos de funcionamiento en cuanto al nivel de seguridad que se ofrezca, es decir, a qué nivel se disparan los procedimientos que garanticen la seguridad.

2.1.10.1 Clasificación de los Dispositivos Según la Perspectiva de Seguridad

- *Dispositivos Confiables:* Mantienen un vínculo de confianza fijo, ya se han autenticado al menos una vez y también se les ha catalogado como “confiable” por parte del usuario del dispositivo al cual el dispositivo confiable quiere acceder; esto le garantiza acceso libre a

todos los servicios. Por ejemplo, para la impresora, un dispositivo de confianza sería el PC que se encuentra fijo en la misma oficina, el cual no necesitará autenticarse cada vez que necesite hacer una impresión; inicialmente el usuario debe haber configurado para aceptar a tal equipo fijo como “confiable”.

- *Dispositivos No Confiables:* No tienen un vínculo permanente, aunque si temporal de seguridad. Se le restringe el acceso a servicios. Para la impresora del ejemplo, un dispositivo no confiable podría ser aquel equipo de una oficina adyacente ó el PC portátil que ocasionalmente entra a la oficina e imprime con permiso del propietario de la impresora.

2.1.10.2 Clasificación de los Servicios Según la Perspectiva de Seguridad

- *Servicios Abiertos:* En esta clase de servicios, todos los dispositivos podrán tener acceso, no hay ninguna barrera de seguridad para ellos.
- *Servicios de Solo Autenticación:* En este caso los servicios se concentran en solicitar autenticación. Si pasa las pruebas de autenticación, se tendrá acceso al servicio.
- *Servicios de Autenticación y Autorización:* Requieren que haya previamente un lazo de confianza con el dispositivo solicitante del servicio. Cualquier otro dispositivo debe solicitar autorización de manera manual (con intervención del usuario). La autorización siempre implica autenticación previa.

En el perfil de acceso genérico la seguridad se garantiza usando dos métodos: un proceso de autenticación y/o una elección del modo de seguridad.

2.1.10.3 Autenticación

El procedimiento genérico de autenticación describe cómo se usan los procedimientos de autenticación a nivel del LMP cuando la autenticación se inicia desde un dispositivo Bluetooth hacia otro, dependiendo de si existe o no una clave de enlace. Cabe anotar que el dispositivo que inicia la autenticación tiene que estar en el Modo de Seguridad 2 o en el Modo de Seguridad 3.

2.1.10.4 Modos de seguridad

2.1.10.4.1 Modo de seguridad 1

Este nivel de seguridad no realiza ningún filtrado ni barrera de seguridad, es decir, nunca inicia procedimientos de autenticación ó encriptación para entablar comunicaciones.

2.1.10.4.2 Modo de seguridad 2

Es un modo de Seguridad más general, puesto que el primero podría considerarse como un caso específico del segundo, donde no se han establecido políticas de seguridad ni requerimientos para los servicios; estos requerimientos pueden combinar factores como Autenticación (para evitar suplantación de identidad de dispositivos), Autorización (Para establecer privilegios) y Encriptación (para proteger la privacidad de la información transportada).

Cuando un dispositivo solicita establecer un canal de comunicación a través del protocolo de control de enlace lógico (L2CAP) es cuando se inician los procedimientos de seguridad. Esto significa que la seguridad es a nivel del servicio. El nivel de permisividad a que podrá llegar otro dispositivo es hasta establecer enlaces a nivel de *Link Manager,* pero el acceso a aplicaciones será restringido por este modo de seguridad.

2.1.10.4.3 Modo de seguridad 3

En este modo la seguridad puede iniciarse a nivel del *Link Manager.* Antes de terminar el establecimiento de un enlace a ese nivel, se debe iniciar algún procedimiento de autenticación, autorización ó encriptación según se requiera.

2.2 PERFIL DEL PUERTO SERIAL

El Perfil del Puerto Serial define los requerimientos necesarios para que dispositivos Bluetooth establezcan conexiones emuladas de cable serial RS232 (o similares) entre dos dispositivos finales a través del nivel RFCOMM. Dichos requerimientos se expresan en términos del servicio ofrecido a la aplicación, y determinan las características y procedimientos que se necesitan para garantizar la interoperabilidad entre dispositivos Bluetooth.

El escenario cubierto por este perfil abarca aplicaciones heredadas que usan Bluetooth para reemplazar cables, a través de una abstracción virtual del puerto serial (la cual es dependiente del sistema operativo).

Algunas aplicaciones manejan información de carácter similar aunque se encuentren en diferentes dispositivos, dichas aplicaciones normalmente necesitan comunicarse para compartir tal información. Para ello se han habilitado dichas aplicaciones con capacidades para utilizar el puerto serial, de tal forma que puedan intercambiar información. Estas capacidades las ofrecen los

manejadores de los puertos seriales cableados, los cuales ofrecen registros, recursos de memoria para datos de entrada y salida, además de una funcionalidad para enviar y recibir datos.

Bluetooth fue creado inicialmente con el propósito de reemplazar los cables de conexión de datos existentes, y los cables seriales no podrían ser una excepción puesto que son los más utilizados. Garantizar que las aplicaciones que utilizan interfaces seriales cableadas puedan operar igualmente sobre Bluetooth es una característica casi necesaria para que esta tecnología sea una alternativa viable a la hora de elegir el tipo de interfaz de comunicaciones en el diseño de equipos y de nuevos productos y aplicaciones.

2.2.1 Stack de Protocolos del Perfil

La Figura 2.2 muestra los protocolos y entidades usados en este perfil:

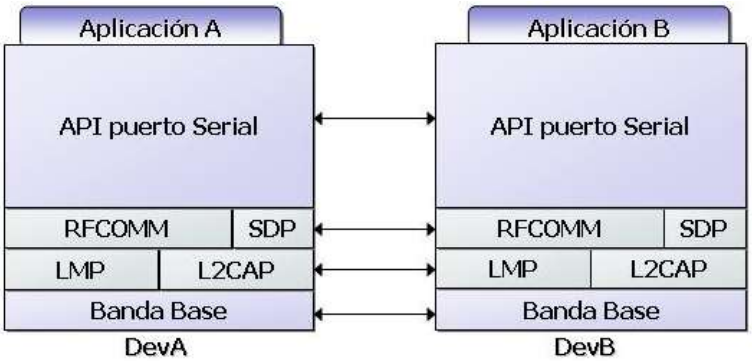


Figura 2.2 Modelo de Protocolo

Banda base, LMP y L2CAP son protocolos de las capas Bluetooth correspondientes a los niveles físico y de enlace en el modelo de referencia OSI. RFCOMM proporciona un protocolo de transporte para la emulación del puerto serial. SDP es el Protocolo de Descubrimiento del Servicio Bluetooth.

La capa de emulación del puerto es una entidad que emula el puerto serial, o proporciona una API a las aplicaciones. Las aplicaciones, en ambos casos están habilitadas y quieren establecer una comunicación sobre un cable serial. Pero hasta este punto, las aplicaciones no tienen conocimiento acerca de los procedimientos Bluetooth para establecer una emulación de cable de puerto serial, es por esto que se requiere de una aplicación Bluetooth adicional en ambos lados. En este perfil; la mayor preocupación es garantizar la interoperabilidad Bluetooth.

2.2.2 Configuración de los Roles

Los siguientes roles están definidos para este perfil:

- Dispositivo A (**DevA**) - Este es el dispositivo que toma la iniciativa para establecer una conexión con otro dispositivo (DevA es el llamado “Parte A” descrito en la sección *Configuración de los Roles* expuesta en el Perfil de Acceso Genérico - GAP).
- Dispositivo B (**DevB**) - Este dispositivo espera que otro dispositivo tome la iniciativa de conectarse. (DevB es el llamado “Parte B” descrito en la sección *Configuración de los Roles* expuesta en el Perfil de Acceso Genérico - GAP).

Es de anotar que el orden de conexión (desde DevA hacia DevB) no necesariamente implica el orden en que las aplicaciones se inician en cada lado respectivamente.

2.2.3 Requerimientos de Usuario y Escenarios del Perfil

El escenario cubierto por este perfil es el siguiente:
Establecer puertos seriales virtuales (o equivalentes) en dos dispositivos (por ejemplo PCs), conectarlos usando Bluetooth y emular un cable de puerto serial entre los dos. Cualquier aplicación puede ejecutarse en alguno de los dos dispositivos, usando el puerto serial virtual como si hubiera un cable serial real que los conectara (con señalización de control RS232). Solo se permite una conexión al tiempo. Por consiguiente, se consideran sólo configuraciones punto a punto. Sin embargo, esto no debe verse como una limitación, ya que en un mismo dispositivo pueden estar corriendo múltiples aplicaciones de este perfil.

2.2.4 Fundamentos del perfil

Para la ejecución de este perfil, es opcional el uso de características de seguridad como autorización, autenticación y encriptación. Si se desean usar las características de seguridad, los dos dispositivos deben hacerlo durante la fase de establecimiento de la conexión.

- El establecimiento del enlace lo inicia el DevA. Entonces, los procedimientos de descubrimiento del servicio deben realizarse para establecer una conexión de cable serial emulada.
- No están establecidos los roles de maestro/esclavo.
- RFCOMM se usa para transportar los datos del usuario, señales de control del módem y comandos de configuración.

2.2.5 Carácter Serial de algunas Aplicaciones

Los ejemplos de aplicaciones seriales son muy diversos, las cámaras, los dispositivos de almacenamiento externos al computador, las transferencias de archivos y aplicaciones de sincronización de agenda y calendario son una muestra de ello. El carácter serial de las aplicaciones presente en aquellas como intercambio de objetos, envío de objetos, transferencia de archivos, sincronización y acceso a redes, es prácticamente evidente, pero lo que no es evidente es que tales aplicaciones generalmente requieren de la presencia de un cliente y servidor escuchando y transmitiendo sobre la interfaz serial; esto conlleva a que cada una de las aplicaciones debería tener a su disposición una interfaz de este tipo; de ahí que se requiera una forma de multiplexar los múltiples puertos seriales que se necesiten, capacidad que ofrece la capa RFCOMM de la pila de protocolos.

2.2.5.1 Equivalencia entre una Aplicación Serial Cableada y una Aplicación Serial Inalámbrica

Una aplicación serial puede ser por ejemplo, la utilización de un cable serial para conectar dos computadores personales (en este caso, se utiliza la configuración de cable cruzado ó *Null Módem*[11]), en tal caso uno de los dos equipos debe iniciar la comunicación sabiendo de antemano que el otro está presente en el otro lado, si esto no ocurre, entonces el iniciador debe anunciar este hecho al usuario. Para que la información pueda ser intercambiada entre los dos dispositivos se debe cumplir que:

- El DevA debe saber de antemano que el otro equipo ya está conectado a la interfaz serial cuando recibe el comando del usuario para "*Conectar con Equipo Remoto*".
- El DevA también debe conocer qué número de puerto COM debe utilizar y a cual está conectado el otro equipo; esto último lo sabe seguramente gracias a una entrada del usuario.

La anterior información se debe proporcionar a los dispositivos cuando la conexión es cableada; en el caso equivalente inalámbrico, la información es esencialmente la misma; para suministrarla y realizar la conexión, la capa del protocolo RFCOMM, el perfil de aplicación serial, en conjunto con el de *Acceso Genérico* y el *Descubrimiento de Servicios* plantean una solución que provee interoperabilidad entre las aplicaciones y que cumplan con ciertos requisitos.

2.2.5.2 Implicaciones de las Aplicaciones Seriales Bluetooth

Las aplicaciones seriales inalámbricas deben seguir varios pasos antes que puedan establecer las conexiones seriales con otros dispositivos:

[11] Configuración de cable serial que consiste en usar un cable serial para periférico, pero cruzando sus terminales de Tx y Rx, también las de RTS y CTS, y las de DTR y DSR. En esta configuración el cable se comporta como un módem visto desde ambos dispositivos, de ahí su nombre.

1. El DevA debe descubrir al otro dispositivo y averiguar si posee las capacidades para soportar servicios que tengan que ver con puertos seriales (por ejemplo, en el caso de un periférico, el parámetro *Class of Device* contiene el código correspondiente a un dispositivo periférico como joystick, teclado ó mouse); para ello puede utilizar la funcionalidad descrita en el perfil de acceso genérico para descubrir el dispositivo, usando alguno de los modos de descubrimiento general ó limitado, y luego obtener información básica acerca de si el dispositivo encontrado es en realidad al que se quiere conectar (seleccionarlo de una lista de dispositivos encontrados). Hasta aquí la funcionalidad necesaria, como se puede ver, se limita a la proporcionada por el perfil de acceso genérico. También se debe tener en cuenta las características de seguridad requeridas, puesto que en algún momento el otro dispositivo puede solicitar autenticación.
2. Luego de tener descubierto el dispositivo, el DevA debe buscar la aplicación dentro del dispositivo remoto y para ello solicita un canal L2CAP para iniciar una sesión de descubrimiento de servicios con el dispositivo remoto; a través de sucesivas solicitudes y respuestas se debe obtener el identificador del canal serial (*Data link Connection Identifier ó* DLCI[12] definido por RFCOMM), es importante resaltar que entonces la aplicación del dispositivo remoto debe haberse registrado en su base de datos de descubrimiento para lograr ser descubierta por el DevA.
3. Una vez obtenido el DLCI, la aplicación del dispositivo iniciador sabe que en éste canal la aplicación remota esta escuchando solicitudes, un concepto muy similar al implementado por los puertos TCP donde de antemano se sabe en qué puerto se encuentra una aplicación específica, es el caso de los puertos “conocidos” como por ejemplo, el 25 para encontrar un servidor de correo, ó el 80 para servidores web, 21 para ftp, etc.
4. El DevA solicita un canal L2CAP con el parámetro PSM[13] establecido en el valor correspondiente a RFCOMM. Esto establece una sesión RFCOMM y una vez establecida, utiliza el DLCI obtenido para abrir el canal serial que comunicará a las dos aplicaciones.

Asumiendo una aplicación que esta soportada por una interfaz serial como por ejemplo algún servidor OBEX (Transferencia de archivos, sincronización y envío de objetos); cada vez que un cliente se quiera conectar a estos servidores y establecer una sesión, debe ocurrir este proceso por parte del cliente, previa iniciación del servidor.

[12] Es un identificador para cada conexión de datos que se establece en una sesión de RFCOMM, es único en una sesión y representa un flujo de datos entre una aplicación Cliente/Servidor.

[13] Protocol and Service Multiplexor – Multiplexor de Servicios y Protocolos; código que identifica el tipo de servicio ó protocolo que va a utilizar el canal L2CAP que se solicita.

2.2.6 Manejo del modo de potencia y pérdida del enlace

Los requerimientos de potencia para cada dispositivo Bluetooth activo en el Perfil de Puerto Serial pueden ser significativamente diferentes, aunque no es obligatorio usar algún modo de potencia, se requiere usar un modo de baja potencia (Sniff, Hold, Park).

Si un dispositivo detecta la pérdida del enlace, a nivel de RFCOMM se debe considerar una interrupción. Antes de que se pueda reanudar con la comunicación en los niveles superiores, se debe ejecutar el procedimiento de iniciar sesión RFCOMM.

2.3 PERFIL APLICACIÓN DE DESCUBRIMIENTO DEL SERVICIO

El Perfil de Aplicación de Descubrimiento del Servicio (Service Discovery Aplication Profile – SDAP) define los protocolos y procedimientos que deben ser implementados por una aplicación, para localizar servicios en otros dispositivos utilizando el protocolo para descubrimiento de servicios (Service Discovery Protocol - SDP) que proporciona el stack de protocolos de Bluetooth, además este perfil describe un modelo de aplicación general y define las abstracciones de las primitivas de servicio que permiten la creación de Interfaces de Programación de Aplicaciones (Application Programming Interface - APIs). SDAP define el modo en que otros perfiles de aplicación, deben utilizar los protocolos de transporte de Bluetooth.

2.3.1 Apreciación del perfil

El perfil aplicación de descubrimiento del servicio es una aplicación específica iniciada por el usuario. En este perfil es dónde las interacciones del descubrimiento del servicio entre dos entidades SDP en dos dispositivos habilitados con Bluetooth, resultan de la necesidad de habilitar un servicio de transporte o modelo de uso en particular (por ejemplo transferencia de archivos, telefonía inalámbrica, LAN, etc.) sobre estos dos dispositivos.

2.3.2 Stack de Protocolos del Perfil

La Figura 2.3 muestra la arquitectura de protocolos Bluetooth y entidades involucradas en el perfil para descubrimiento de servicios.

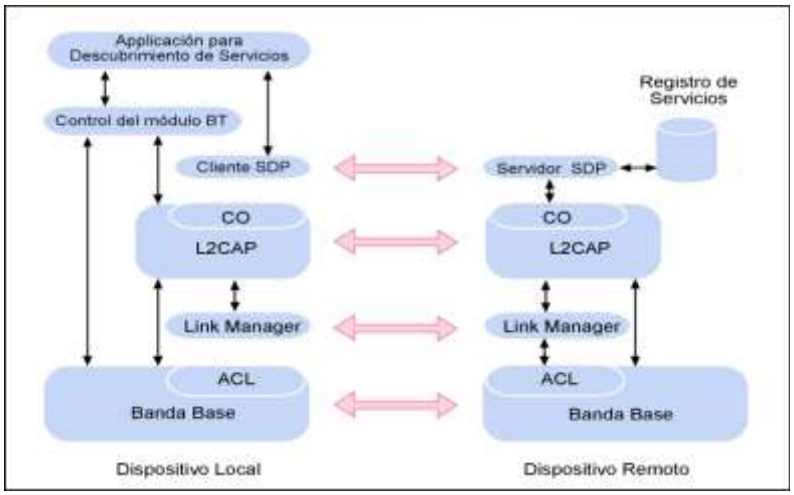


Figura 2.3 Arquitectura de protocolos y entidades que componen el SDAP

La aplicación del usuario para el descubrimiento del servicio (SrvDscApp) que se encuentra alojada en un dispositivo local (LocDev) mediante una interfaz se comunica con un cliente SDP Bluetooth para enviar las indagaciones de servicio y recibir las respuestas a dichas indagaciones de servicio desde los servidores SDP que se encuentran en los dispositivos remotos (RemDevs).

El SDP utiliza el servicio de transporte orientado a la conexión (SCO) en L2CAP, el cual a su vez utiliza en la banda base, un enlace no orientado a la conexión asíncrono (ACL) para finalmente transportar los SDP PDUs[14] sobre el aire. El descubrimiento de servicio esta relacionado con el propósito de descubrir dispositivos, y el descubrimiento de dispositivos a su vez esta relacionado con realizar indagaciones y paginaciones. Así, las interfaces SrvDscApp con la banda base a través de la entidad de Control de Modulo Bluetooth (BT_module_Cntrl[15]) le dicen al módulo Bluetooth cuándo entrar en alguno modo de búsqueda del funcionamiento. La base de datos del registro de servicio mostrada en la Figura 2.3 es una entidad lógica que sirve como almacén de información relacionada con el descubrimiento del servicio.

2.3.3 Configuración de los Roles

- Dispositivo local (LocDev): Un LocDev es el dispositivo que inicia el procedimiento de Descubrimiento del Servicio, este debe contener al menos la parte de la arquitectura del cliente SDP Bluetooth, además de la aplicación de descubrimiento del servicio

[14] Protocol Data Unit – Unidad de Datos de Protocolo
[15] El Control de Modulo Bluetooth - BT_module_Cntrl, puede formar parte de la implementación del Stack de protocolos de Bluetooth o una pequeña parte de la aplicación del Descubrimiento del Servicio.

(SrvDscApp) usada por un usuario para iniciar los procesos de descubrimiento y desplegar los resultados de dichos descubrimientos.

- Dispositivo remoto (RemDev(s)): Un RemDev es cualquier dispositivo que participa en el proceso de descubrimiento del servicio respondiendo a las indagaciones de servicio generadas por un LocDev. este debe contener al menos la parte de la arquitectura del servidor SDP Bluetooth. Además contiene una base de datos del registro de servicio, la cual es consultada por el servidor SDP para generar respuestas a las solicitudes del descubrimiento del servicio.

Un dispositivo Remoto puede clasificarse de acuerdo con el grado de confianza que represente para un dispositivo local, así:

Dispositivo de confianza: Son dispositivos que en el momento no se encuentran conectados al dispositivo local, pero guardan una relación de confianza con este.

Dispositivos Desconocidos: (Nuevos dispositivos) Son dispositivos que jamás han establecido conexión con el dispositivo local.

Dispositivos Conectados: Son dispositivos que se encuentran conectados a la red y con los cuales el dispositivo local ha establecido un enlace.

El papel de un dispositivo remoto o dispositivo local no es ni permanente ni exclusivo para cierto dispositivo, el cual podrá asumir cualquiera de los dos; e incluso al mismo tiempo, según se presenten las circunstancias de comunicación entre dos dispositivos Bluetooth. Un RemDev puede tener una SrvDscApp instalada así como un cliente SDP, y un LocDev puede tener instalado un servidor SDP. Para un dispositivo que tenga instalado la Aplicación SrvDscApp, con el cliente y el servidor SDP, la asignación de los roles anteriores, es relativa a cada transacción SDP individual y relacionada a qué dispositivo comienza la transacción. Así, un dispositivo podría ser un LocDev para una transacción SDP en particular, mientras tanto, al mismo tiempo ser un RemDev para otra transacción SDP.

2.3.4 Requerimientos de Usuario y Escenarios del Perfil

Los Escenarios de Aplicación de este perfil son los siguientes:

- Búsqueda de servicios según la clase de servicio.
- Búsqueda de servicios según los atributos de servicio.
- Navegador de servicios (Browsing de servicios).

Los primeros dos casos representan búsquedas de servicios conocidos y específicos, como parte de la pregunta del usuario "Esta el servicio A, o esta el servicio A con características de B y C, disponible?" El ultimo caso representa una búsqueda de servicio general que es una respuesta a la pregunta del usuario "Qué servicios están disponibles?".

2.3.5 Fundamentos del perfil

Antes de que dos dispositivos habilitados con Bluetooth puedan comunicarse entre sí, es necesario que cumplan lo siguiente:

- Los dispositivos necesitan estar encendidos e inicializados. La inicialización puede requerir proporcionarle un PIN para la creación de una clave de enlace, para la autorización del dispositivo y encriptación de los datos.
- Se tiene que crear un enlace Bluetooth, el cual puede requerir el descubrimiento de la dirección del otro dispositivo Bluetooth BD_ADDR a través de un proceso de indagación y paginación.

El descubrimiento de servicio puede iniciarlo cualquier dispositivo maestro o esclavo en algún instante durante el cual estos dispositivos pertenezcan a la misma piconet. También, un esclavo en una piconet posiblemente pueda comenzar el descubrimiento de servicio en una nueva piconet, siempre y cuando le notifique al maestro de la piconet inicial que no estará disponible por un tiempo dado (posiblemente entrando al modo de funcionamiento hold).[16]

2.3.6 Aspectos Relacionados a la Interfaz de usuario

- *Selección del modo*: Este perfil asume que bajo la supervisión de la SrvDscApp, el LocDev podrá entrar en el estado indagación y/o de paginación. También se asume que un RemDev con servicios que están disponibles a otros dispositivos (por ejemplo una impresora, LAN, gateway PSTN, etc.) puede entrar en estados que puedan ser descubiertos y /o paginados. Siempre y cuando la SrvDscApp haya realizado las consultas del servicio en un RemDev ya conectado, no es obligatorio, que el LocDev sea siempre el maestro de la conexión. De manera similar, un RemDev no debe ser siempre el esclavo de la misma conexión.

[16] Es importante tener en cuenta que un maestro de una Piconet, no puede iniciar una nueva Piconet. Puesto que la Piconet esta identificada por la Dirección del Dispositivo Bluetooth – BD_ADDR y el reloj del dispositivo maestro.

2.3.7 Capa de aplicación

2.3.7.1 Aplicación de Descubrimiento del Servicio

En la Figura 2.4 se presenta la forma en que puede implementarse la estructura de operación de una Aplicación para Descubrimiento de Servicios *SrvDscApp*.

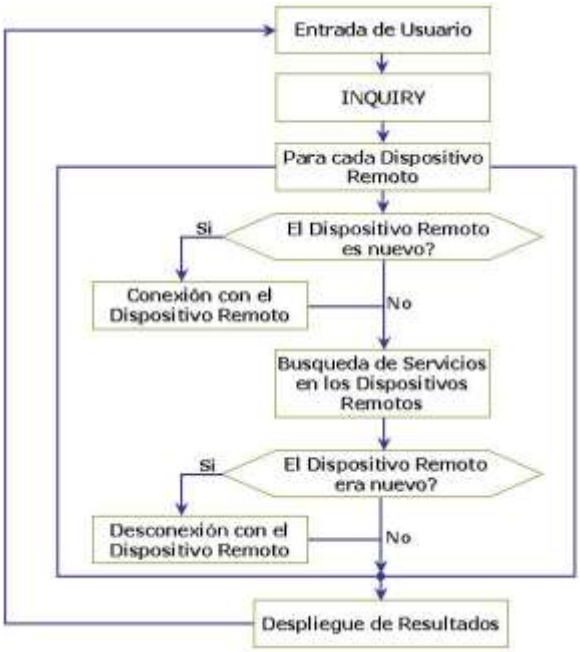


Figura 2.4 Estructura de Operación de la SrvDscApp

1. La Aplicación para Descubrimiento de Servicios *SrvDscApp* activa una operación de INQUIRY o INDAGACION, seguida de una petición del usuario para la búsqueda de servicios.
2. Para cualquier Dispositivo Remoto que se encuentre después del INQUIRY se ejecutan las operaciones para descubrimiento de servicios y una vez estas terminen, el Dispositivo Local finaliza su enlace con el dispositivo remoto y está en capacidad de conectarse con un nuevo dispositivo.
3. Si antes de realizar el INQUIRY, ya se encontraban conectados otros dispositivos con el Dispositivo Local, este no los desconectará después del descubrimiento de servicios.
4. El usuario de la Aplicación para Descubrimiento de Servicios tiene la opción de conectarse con dispositivos remotos, operando en modos que tienen distintos niveles de *confianza*, así:
    - Con dispositivos remotos confiables, solamente.
    - Con Dispositivos remotos confiables, mas dispositivos remotos que solo requieren la entrada de un PIN de valor cero.

- Con cualquiera de los dispositivos anteriores, más cualquier Dispositivo Remoto adicional que requiera la entrada de un PIN diferente de cero, por parte del usuario.

Como se explico anteriormente cuando un LocDev realiza una búsqueda de descubrimiento del servicio, lo hace como si fueran tres tipos diferentes de RemDevs:

1. Dispositivo de confianza
2. Dispositivos Desconocidos
3. Dispositivos Conectados

Para descubrir RemDevs de tipo 1 o 2, la SrvDscApp necesita activar los procesos indagación y paginación. Para RemDevs de tipo 3, es necesario solo el proceso de paginación. Para realizar esta tarea, la SrvDscApp necesita tener acceso a la BD_ADDR de los dispositivos en la vecindad del LocDev, no importa si estos dispositivos han sido localizados a través de un proceso de Indagación Bluetooth o se han conectado al LocDev. Así, el BT_module_Cntr en un LocDev deberá mantener la lista de dispositivos en la vecindad del LocDev y esta lista debe serle útil a la SrvDscApp.

2.3.7.2 Abstracciones de las Primitivas del Servicio

La funcionalidad de una *SrvDscApp* se presenta en forma de abstracción de las primitivas de servicio. Esta abstracción proporciona una estructura formal que describe las expectativas que tiene el usuario de la *SrvDscApp*. La sintaxis y semántica exactas de las abstracciones de las primitivas del servicio (o simplemente "primitivas del servicio") pueden ser dependientes de la plataforma ya sea un sistema operativo, una plataforma hardware, como una PDA, una agenda digital, un teléfono celular sin embargo esta parte está más allá del alcance del perfil. Sin embargo, la funcionalidad que representan estas primitivas deben estar disponibles en la SrvDscApp para cumplir correctamente su tarea.

La Tabla 2.1 describe las abstracciones de las primitivas de servicio requeridas por el perfil.

PRIMITIVA	RESULTADO
Búsqueda de Servicios Service Browse	Se realiza un Búsqueda de servicios contenidos en un Grupo de Búsqueda en los servicios que hacen parte de una lista de dispositivos remotos. Además, la búsqueda puede realizarse teniendo en cuenta una lista de parámetros que indican la relación de confianza o conexión que un dispositivo local

	mantiene con un dispositivo remoto (RemDevRelation). Por ejemplo una búsqueda solo de dispositivos que se encuentren en la lista de dispositivos remotos para los cuales ya se tiene una relación de confianza establecida, entonces cuando ya se obtiene el nombre de los RemDevs que soportan los servicios solicitados estos son retornados; la búsqueda continúa hasta que se hace efectiva, una regla de detenimiento de la acción.
Búsqueda de Servicios ServiceSearch	Se realiza una búsqueda de servicios con el fin de saber si un dispositivo que se encuentra en la lista de dispositivos remotos, soporta los servicios coincidentes con un patrón de búsqueda y una lista de Atributos. En este caso, la búsqueda también puede calificarse con una lista de parámetros que indican la relación que un dispositivo local mantiene con un dispositivo remoto (RemDevRelation). Cuando ya se obtiene el nombre de los RemDevs que soportan los servicios solicitados estos son retornados; la búsqueda continúa hasta que se hace efectiva, una regla de detenimiento de la acción.
Enumeración de Dispositivos Remotos EnumerateRemDev	Ejecuta una búsqueda de dispositivos remotos en la vecindad del dispositivo local. La búsqueda puede filtrarse ocasionalmente utilizando la lista de clases de dispositivo. La búsqueda continúa hasta que se hace efectiva, una regla de detenimiento de la acción.
Primitiva de Terminación TerminatePrimitive	Se presenta un término para que finalicen las acciones que se llevan a cabo.

Tabla 2.1. Primitivas del Servicio para SDAP

Las primitivas de servicio anteriores devuelven la información solicitada, siempre y cuando la encuentren. Los resultados de una búsqueda pueden regresar directamente por el correspondiente llamado a la función, o por medio de un puntero a la estructura de datos

La regla de detenimiento se emplea para garantizar la culminación de una búsqueda de servicios y esta podría representar el número total de ítems que se encontraron, así como la duración de la búsqueda.

2.3.7.3 Diagrama de Secuencia de mensajes

El perfil para el descubrimiento de servicios involucra tres procedimientos Bluetooth diferentes que son: descubrimiento del dispositivo, descubrimiento del nombre del dispositivo y descubrimiento del servicio. Cada uno de estos procedimientos no impide el otro; por ejemplo para conectarse a un RemDev, un LocDev puede primero descubrirlo, y también puede preguntar su nombre.

La Figura 2.5 resume las "fases" de intercambio de mensajes claves que se dan durante la ejecución de este perfil. No todos los procedimientos están presentes en todo momento, y no todos los dispositivos necesitan pasar por estos procedimientos. Por ejemplo, si no se requiere de autenticación, la fase de autenticación en la Figura 2.5 no se ejecutará. Si la SrvDsvApp necesita indagar por servicios en un RemDev específico con el cual el LocDev ya se ha conectado anteriormente, puede que no sea necesario ejecutar los procedimientos de indagación (inquiry) y paginación (paging).

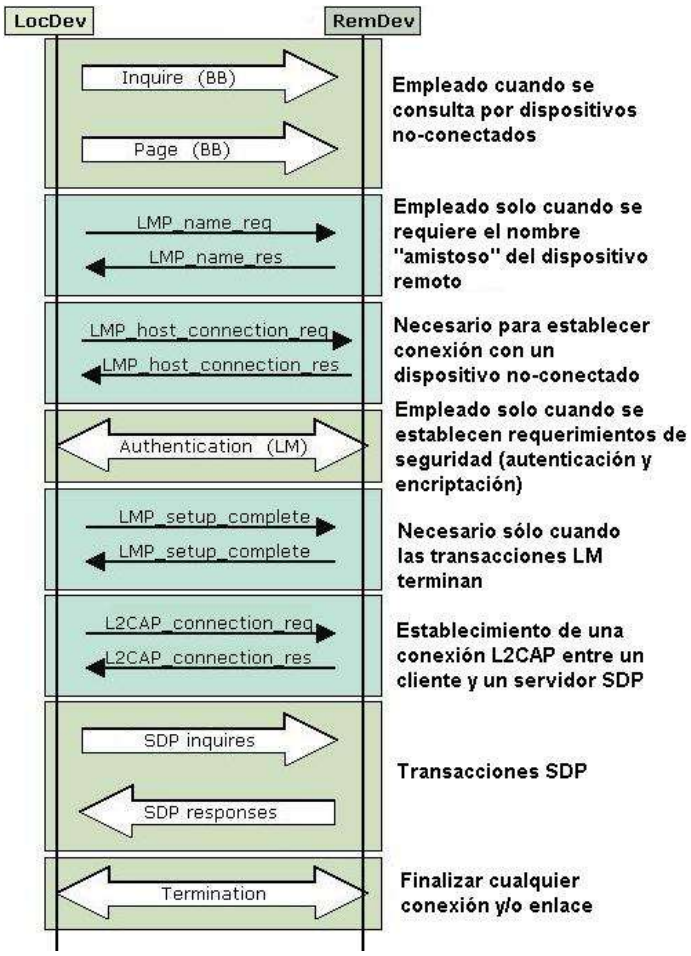


Figura 2.5 Intercambio de mensajes durante el SDAP

2.4 PERFIL DE ACCESO A LAN

El perfil de Acceso a una LAN le permite a un dispositivo habilitado con Bluetooth obtener acceso a una red de datos, y en lugar de utilizar un teléfono o un módem, emplear un Punto de Acceso a datos conectado a la red externa.

Usar el perfil de Acceso a una LAN, es similar a establecer una conexión con una red mediante un cable Ethernet, pero de manera inalámbrica restringiéndose al empleo del protocolo Punto a Punto, sobre RFCOMM.

Las razones por las cuales se escogió este protocolo son:

- PPP es una de las formas más empleadas para permitir el acceso a redes.
- PPP ofrece mecanismos de autenticación, encriptación, compresión de datos y soporte multi-protocolo.
- PPP permite que muchos dispositivos, incluyendo las PDA's soporten comunicación IP sobre PPP para marcar a redes IP.

El Perfil de Acceso a una LAN para dispositivos Bluetooth se divide en dos partes:

- La primera parte define la manera en que los dispositivos habilitados con Bluetooth pueden acceder a los servicios de una LAN usando PPP.
- La segunda parte muestra cómo los mismos mecanismos PPP se utilizan para formar una red que consiste en dos dispositivos habilitados con Bluetooth.

Básicamente este perfil define el acceso a una LAN usando PPP sobre RFCOMM (En el futuro pueden aparecer otras maneras y medios de acceder a una LAN).

2.4.1 Apreciación del perfil

Este perfil define cómo las conexiones PPP se soportan en las siguientes situaciones:

- Acceso a una LAN para un sólo dispositivo Bluetooth.
- Acceso a una LAN para múltiples dispositivos Bluetooth.
- PC a PC (usando conexiones PPP sobre la emulación de cable serial).

2.4.2 Stack de Protocolos del Perfil

La Figura 2.6 muestra la pila de protocolos y entidades para implementar el perfil de acceso a LAN. Además de las capas convencionales de Bluetooth, se incluyen las siguientes capas adicionales:

- PPP: Protocolo Punto a Punto.
- Intertrabajo o *Networking PPP*: Esta capa establece la forma en que se toman paquetes IP de la capa PPP para introducirlos en la LAN.
- Entidad de Gestión *Management Entity* ME: Se encarga de coordinar los procedimientos de inicialización, configuración y gestión de la conexión.

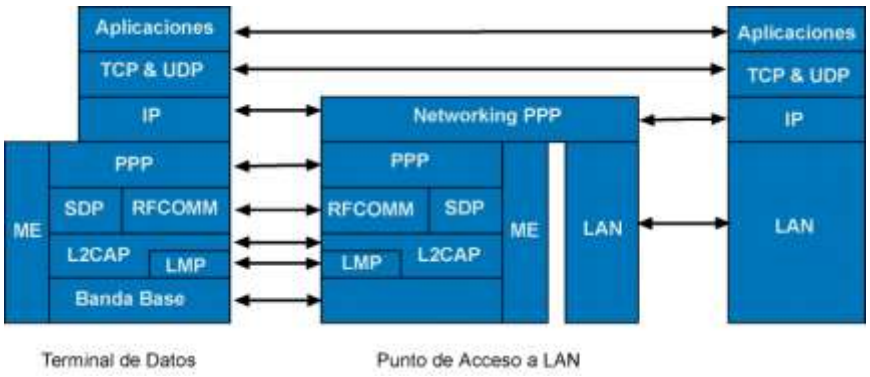


Figura 2.6 Pila de protocolos para el perfil de Acceso a LAN

2.4.3 Configuración de los Roles

En este perfil se definen dos roles para los dispositivos Bluetooth:

- **Punto de Acceso a LAN (LAP)** – Hace referencia al dispositivo Bluetooth que proporciona el acceso a una LAN. El LAP proporciona los servicios de un Servidor PPP y la conexión se soporta sobre RFCOMM. RFCOMM se usa para transportar los paquetes PPP y también puede usarse para el control del flujo de los mismos.
- **Terminal de Datos (DT)** - Éste es el dispositivo que usa los servicios del LAP. Dispositivos típicos que actúan como terminales de datos son computadoras portátiles, PDAs, PCs de escritorio. El DT actúa como un cliente PPP.

2.4.4 Requerimientos de Usuario y Escenarios del Perfil

Los siguientes escenarios son cubiertos por este perfil:

- Acceso a una LAN por un sólo PC: En este escenario un sólo DT usa un LAP como un medio inalámbrico para conectarse a una LAN. Una vez conectado, el DT operará como si

se conectara a la LAN a través de una conexión por línea conmutada. El DT puede acceder a todos los servicios proporcionados por la LAN. En la Figura 2.7 se puede observar este escenario.

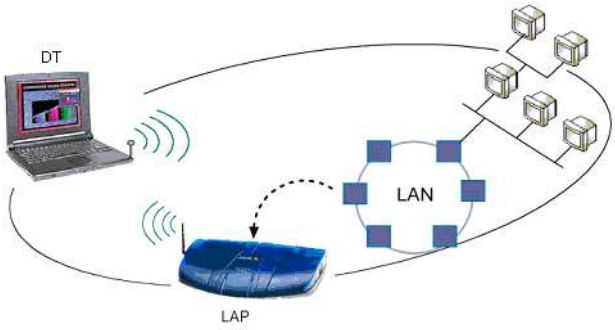


Figura 2.7 Escenario Acceso a una LAN por un sólo DT

- Acceso a una LAN por múltiples DTs: Aquí, muchos DTs usan un LAP como el medio inalámbrico para conectarse a una Red de Area Local (LAN). Una vez conectado, los DTs operarán como si se conectaran a la LAN a través de una conexión por línea conmutada. El DTs puede acceder a todos los servicios proporcionados por la LAN. En la Figura 2.8 se puede observar este escenario.

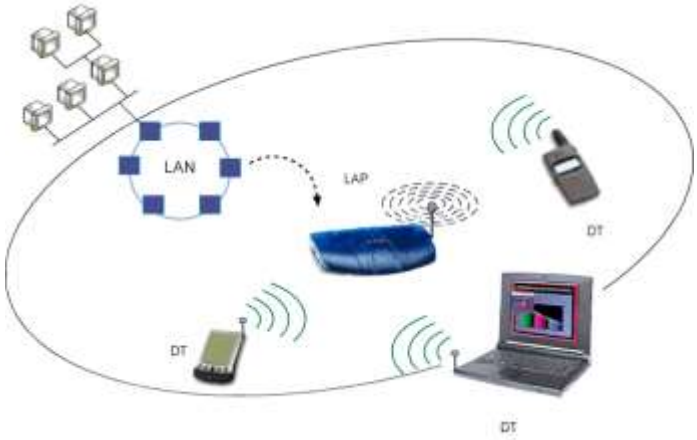


Figura 2.8 Escenario Acceso a una LAN por múltiples DTs

- Conexión PC a PC: Dos dispositivos Bluetooth pueden formar una sola conexión entre sí. Este escenario normalmente es similar a una conexión de cable directa comúnmente usada para conectar dos PCs. En este escenario, uno de los dispositivos tomará el papel del LAP, y el otro será un DT. En la Figura 2.9 se puede observar este escenario.

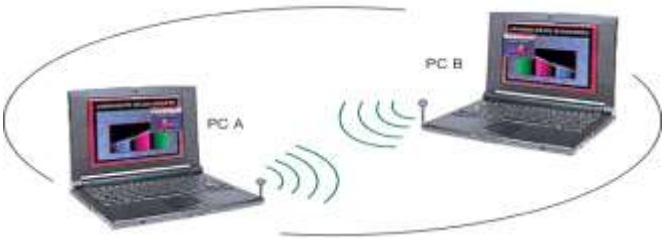


Figura 2.9 Conexión PC a PC

2.4.5 Fundamentos del Perfil

A continuación se describen brevemente, las interacciones que tienen lugar entre dos dispositivos que operan como LAP y DT en el perfil de Acceso a LAN.

1. En primer lugar el DT emplea un tipo de aplicación para encontrar un LAP en el área de cobertura de su radio Bluetooth, con el fin de hacer uso de los servicios de los que dispone.
2. Si no existe un enlace físico a nivel de banda base, entonces el DT debe establecer este enlace a través de una petición de conexión al LAP seleccionado. En algún momento después del establecimiento del enlace físico, los dispositivos realizan la autenticación.
3. El DT establece una conexión PPP /RFCOMM / L2CAP.
4. De manera opcional, el LAP puede usar algún mecanismo de autenticación PPP apropiado, para solicitar la introducción de un PIN, un nombre de usuario y/o la clave de acceso a la red. Por ejemplo, el LAP puede solicitar al usuario del DT que se autentique, entonces el DT debe proporcionar un nombre de usuario y contraseña. Si al hacer esta solicitud el DT no se autentica o se autentica equivocadamente el enlace PPP se caerá.
5. Se negocia una dirección IP apropiada entre el LAP y el DT haciendo uso de procedimientos característicos de PPP.
6. Una vez la conexión PPP se ha establecido; el tráfico IP puede fluir a través de esta conexión.
7. En cualquier instante, el DT o el LAP pueden terminar la conexión PPP.

2.4.6 Aspectos Relacionados a la Interfaz de usuario

Cabe anotar que la construcción de éste perfil se basa en la definición del Perfil de Acceso Genérico. Cuando en el perfil de Acceso Genérico hablamos de DevA y DevB se hace equivalencia con los términos DT y LAP respectivamente dentro del perfil de acceso a LAN.

2.4.6.1 Seguridad

Como la seguridad en un ambiente inalámbrico es de suma importancia, deben tenerse en cuenta las medidas adecuadas para obtenerla. Por lo tanto el LAP y el DT deben forzar a que los procedimientos de encriptación estén operando en el enlace físico establecido mientras cualquier tráfico PPP sea enviado o recibido y además negar cualquier solicitud para desactivar dichos procedimientos.

2.4.6.2 Número de Usuarios

Según el fabricante y la aplicación, un dispositivo Bluetooth puede contar con capacidades diferentes y recursos limitados; lo cual restringe el número de usuarios que pueden hacer uso de él simultáneamente. El LAP debe proporcionar las capacidades para que pueda configurarse en uno de los siguientes modos de operación:

- Modo de Usuario Simple: En este caso el LAP puede ser accedido por un sólo usuario a la vez. En este modo de operación tanto el DT como el LAP pueden asumir el papel de maestro de la Piconet.
- Modo de Usuario Múltiple: En este caso múltiples usuarios pueden acceder a los servicios del LAP de forma simultánea. En esta ocasión, es obligatorio que el LAP se establezca como maestro de la Piconet. Le corresponde a la entidad de Gestión, asegurar que los roles se tomen de esta forma.

Actualmente hay limitaciones prácticas acerca de la cantidad de usuarios que usan una unidad Bluetooth. Cuando hay pocos usuarios simultáneos usando la radio Bluetooth, tendrán disponible más ancho de banda para cada uno.

2.4.7 Capa de Aplicación

2.4.7.1 Inicialización del Servicio Punto de Acceso a LAN

Este procedimiento inicia la configuración del dispositivo como un LAP. Esta operación implica la configuración de los siguientes parámetros:

- Todos los parámetros configurables:
  - Número máximo de usuarios
  - El modo de descubrimiento (Acceso Genérico)
- Los PINs Bluetooth requeridos y/o claves del enlace.
- Cualquier opción de configuración PPP apropiada (por ejemplo la autenticación, compresión).

Cuando se completa la operación de configuración, los dispositivos están listos para aceptar conexiones PPP. Para aquellos dispositivos que están destinados únicamente a la funcionalidad de LAP, los procedimientos anteriores generalmente se llevan a cabo en momento en que se enciende el dispositivo.

2.4.7.2 Suspensión del servicio de Acceso a LAN

Este procedimiento detiene la operación del LAP. En primer lugar se apaga el servidor PPP. Todos los enlaces PPP se desconectan y se remueven todas las entradas de servicio PPP de la base de datos del servidor SDP. El dispositivo puede ejecutar operaciones adicionales para eliminar la información del usuario, con el fin de evitar accesos no autorizados al servicio.

2.4.7.3 Establecimiento de la conexión a LAN

Normalmente el DT iniciará el establecimiento de una conexión a la LAN.

1. El primer paso es seleccionar el LAP que desea utilizarse y un servicio PPP/RFCOMM proporcionado que sea adecuado. Esta selección puede hacerse de alguna de las siguientes maneras:
    - Al usuario del DT se le presenta una lista de LAPs que están dentro del rango de operación, además de los servicios que éstos proporcionan. El usuario puede entonces seleccionar un LAP/servicio con los cuales desea trabajar.
    - Al usuario del DT se le presenta una lista de los servicios que están siendo proporcionados por los LAPs que están dentro del área de cobertura del terminal de datos. Cuando el mismo servicio se proporciona por múltiples LAPs, la aplicación puede elegir mostrar solo una vez el servicio. Entonces el usuario debe seleccionar un servicio de la lista proporcionada y posteriormente el DT seleccionará automáticamente un LAP que presente las mejores condiciones de prestación de ese servicio.
    - El usuario del DT introduce el nombre del servicio que requiere. El DT seleccionará automáticamente un LAP conveniente que proporcione el servicio requerido.
    - Alguna aplicación en el DT busca y selecciona automáticamente servicios y LAPs que se encuentren disponibles en el momento.

En cualquiera de los mecanismos usados, el resultado del proceso de selección debe ser un LAP que esté dentro del rango de cobertura del DT y un servicio que sea proporcionado por el mismo.

En los casos anteriores, se emplean los mecanismos de descubrimiento de servicios del SDP para adquirir la información que desea presentarse al usuario.

2. Puede requerirse la introducción de un PIN por parte del usuario. En caso contrario se emplea un PIN de longitud cero para efectos de conexión.

3. Al usuario DT (o aplicación) se le permite entrar un nombre de usuario y contraseña para la autenticación PPP.

4. Cuando el usuario (o aplicación) activa la conexión, entonces se inicia una aplicación PPP, que intenta establecer una conexión al punto/servicio de acceso seleccionado.

2.4.7.4 Pérdida de conexión con la LAN

Si se pierde conexión con la LAN, el LAP debe notificar al terminal y este a su vez al usuario acerca de la falla en la conexión. Opcionalmente, la aplicación puede permitir el restablecimiento de la conexión a través de la reutilización de los parámetros de configuración y datos de la conexión que acaba de terminarse.

Las siguientes razones causarán la terminación de la conexión:

1. Intervención del usuario.
2. Falla de la conexión RFCOMM/L2CAP, por ejemplo cuando el enlace de radio falla o el dispositivo se sale del rango de cobertura de la Piconet durante una excesiva cantidad de tiempo.
3. Orden de finalización impartida por el LAP cuando este no se encuentra en capacidad de seguir prestando el servicio.
4. Alguna política dependiente de la Aplicación puede causar también la terminación de la comunicación.

2.4.7.5 Desconexión del Enlace

Tanto el LAP como el DT pueden dar término a la conexión en el momento que deseen.

2.4.7.6 Negación del Acceso por presencia de errores en la implementación del perfil

El LAP debe negar al DT el acceso si se presenta una de las siguientes situaciones.

- Hay falla en el proceso de Autenticación
- Ausencia de soporte para encriptación
- Ausencia de soporte de los procedimientos para cambio de rol.

El LAP debe rechazar cualquier intento por parte del DT de realizar los siguientes procedimientos.

- Realizar peticiones para que la encriptación se deshabite.
- Realizar peticiones para que el LAP que opera como maestro, ejecute un cambio de rol si está trabajando en el modo de usuario múltiple. Cuando el LAP se configura en el modo de usuario simple (es decir, el número máximo de usuarios es 1), entonces el LAP puede ser el maestro o esclavo de la piconet. Sin embargo cuando se configura el modo de usuario múltiple (es decir el número máximo de usuarios es más de 1) el LAP debe ser el maestro de la piconet.
- Realizar peticiones para establecer una nueva conexión, cuando se han establecido el máximo número de conexiones permitidas por el valor de Número de Usuarios máximo.

CAPÍTULO 3 ESTUDIO COMPARATIVO DE SOLUCIONES PARA BLUETOOTH

Alrededor de una tecnología en desarrollo generalmente se ubican empresas de diversa índole, propósitos, estudios y pronósticos con el fin de hacer que tal desarrollo sea beneficioso para ellas; es por esta iniciativa que a partir de algunas empresas han surgido herramientas o especificaciones de APIs Bluetooth soportadas en diferentes plataformas tales como Visual C, C++ o Java; las cuales permiten explorar las posibilidades que la tecnología ofrece. Las APIs representan una forma sencilla de programar de acuerdo a las especificaciones Bluetooth siguiendo una abstracción de alto nivel. Este capitulo realiza un estudio de algunas especificaciones.

3.1 API'S JAVA PARA LA TECNOLOGÍA INALÁMBRICA BLUETOOTH JSR-82 ESPECIFICACIÓN VERSIÓN 1.0A

3.1.1 Introducción

Esta API es el resultado de una investigación realizada por el grupo Java Specification Request (JSR-82), en la cual se define un paquete de desarrollo para la tecnología inalámbrica Bluetooth basado en la plataforma 2 de Java Micro-Edición (J2ME). El objetivo de esta especificación es definir la arquitectura y las API's asociadas para permitir un entorno de desarrollo abierto para nuevas aplicaciones Bluetooth.

3.1.2 Grupo JSR-82

Las siguientes compañías son miembros del grupo experto que desarrolló la especificación JSR-82:

Extended Systems	Sharp Laboratories of America
IBM	Sony Ericsson Mobile Communications
Mitsubishi Electric	Smart Fusion
Motorola (Líder de la especificación)	Smart Network Devices
Newbury Networks	Sun Microsystems
Nokia	Symbian
Parthus Technologies	Telecordia
Research in Motion	Vaultus
Rococo Software	Zucotto

3.1.3 Objetivos de la Especificación JSR-82

El objetivo principal de esta especificación es definir un conjunto de API's estándar que ofrezcan un entorno de desarrollo abierto para la implementación de aplicaciones bajo la tecnología inalámbrica Bluetooth. El enfoque primordial de las API's son los dispositivos con capacidad de procesamiento limitada, poca memoria y que operan con batería. Además, como la especificación Bluetooth se encuentra en un continuo proceso de desarrollo, a medida que se adicionan nuevos perfiles, es necesario tener en cuenta que los nuevos perfiles Bluetooth puedan construirse sobre estas API's usando el lenguaje de programación Java, siempre y cuando el núcleo de la especificación no cambie.

3.1.4 Requerimientos de la Especificación JSR-82

Los requerimientos de esta especificación son:

1. Requiere únicamente librerías CLDC[17] - Connected, Limited Device Configuration.
2. La definición del API OBEX, el cual será mencionado mas adelante, debe ser independiente de los protocolos de Bluetooth.
3. Las aplicaciones pueden usar el API OBEX sin necesidad de usar el API Bluetooth, esto se explica con más detalle en la sección 3.1.5.1.
4. La API debe permitir la ejecución de aplicaciones que desempeñen el papel tanto de cliente como de servidor simultáneamente.

3.1.4.1 Requerimientos del dispositivo

Esta API esta diseñada para operar sobre dispositivos con las siguientes características:

[17] Dentro de la arquitectura de J2ME CLDC se define como un conjunto de API's básicas de Java que definen un entorno generalizado de ejecución soportado por un conjunto de dispositivos, para esta configuración los dispositivos son PDA's, Teléfonos Móviles, Pagers, entre otros.

- Memoria total mínima de 512K disponibles para la plataforma Java 2 (ROM/Flash y RAM). Los requerimientos de memoria para la aplicación son adicionales.
- Hardware de comunicación Bluetooth.
- Implementación de la configuración CLDC.

3.1.5 Alcances de la Especificación JSR-82

Como se ha mencionado con anterioridad la especificación Bluetooth define múltiples capas de protocolos y perfiles, sin embargo dentro de la especificación JSR-82 por razones relacionadas con el propósito de la investigación no se incluyen todos estos; por tanto se enfoca en los siguientes aspectos:

- Transmisión solo de datos (Aunque la tecnología inalámbrica Bluetooth soporta transmisión de datos y de audio)
- Los siguientes protocolos: L2CAP (solo orientado a la conexión), SDP, Protocolo de Intercambio de Objetos (OBEX).
- Los siguientes perfiles: Perfil de Acceso genérico (GAP), Perfil de Aplicación del Descubrimiento del Servicio (SDAP), Perfil de Puerto Serial (SPP), Perfil de Intercambio de Objetos Genérico (GOEP)

La especificación JSR-82 no ofrece soporte para los siguientes aspectos:

1. Transmisión de audio
2. El Protocolo de Control de Telefonía (TCS Binario o TCS-BIN)
3. *Capa de Gestión:* Muchos aspectos que tienen que ver con la gestión de dispositivos son sistemas específicos y en algunos casos resulta difícil estandarizarlos, tales como modos de bajo consumo de potencia, entre otros.
4. *Descarga y almacenamiento de aplicaciones:* Estas características son aplicaciones específicas y por consiguiente no se definen en esta especificación.
5. *Inicio Asíncrono de Aplicaciones:* Estos son métodos por los cuales una aplicación puede ser iniciada asincrónicamente debido a peticiones externas no definidas. Por ejemplo, un servicio no tiene que estar corriendo después de que ha sido registrado, pero podría iniciarse cuando un cliente se conecta a ese servicio.

Teniendo en cuenta los requerimientos generales que puede presentar una aplicación, ésta API ofrece las siguientes capacidades:

1. Registro de servicios
2. Descubrimiento de dispositivos y servicios

3. Establecimiento de conexiones L2CAP y OBEX
4. Seguridad a la hora de llevar a cabo estas capacidades.

3.1.6 Arquitectura de la Especificación JSR-82

Basados en el alcance de ésta especificación la funcionalidad ofrecida por la misma se clasifica en tres categorías, como se puede observar en la Figura 3.1

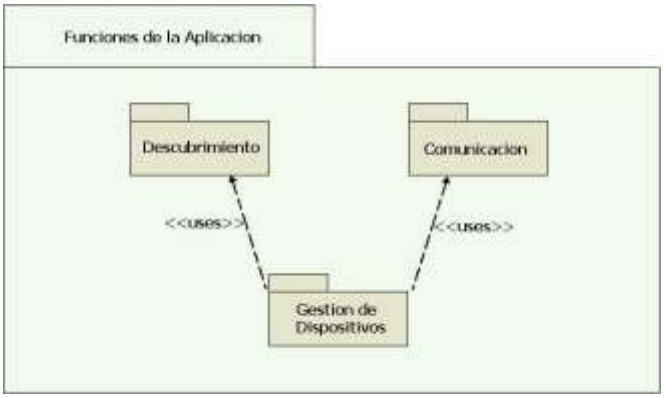


Figura 3.1 Funcionalidad ofrecida por JSR-82

1. *Descubrimiento:* Esta categoría incluye el descubrimiento del dispositivo, descubrimiento del servicio y registro del servicio.
2. *Comunicación:* Incluye establecimiento de conexiones entre dispositivos y el uso de estas conexiones para la comunicación Bluetooth entre las aplicaciones.
3. *Gestión de Dispositivos:* Permite manejar y controlar las conexiones establecidas.

3.1.6.1 Paquetes

Dentro de la especificación se encuentran definidos dos paquetes: el paquete *javax.bluetooth* y el paquete *javax.obex.* Es importante tener claro que la API OBEX esta definida independientemente de la capa de transporte Bluetooth y se empaqueta por separado, esto implica que cualquier aplicación, puede incluir alguno de los dos paquetes o los dos. El primer paquete hace referencia a la API central de Bluetooth y el segundo paquete contiene las API's para OBEX.

La Figura 3.2 muestra la estructura del paquete. Queda claro que tanto el paquete javax.obex como el paquete javax.bluetooth dependen del paquete javax.microedition.io[18]

[18] Paquete estándar de J2ME que se encarga del manejo de los flujos de entrada y salida.

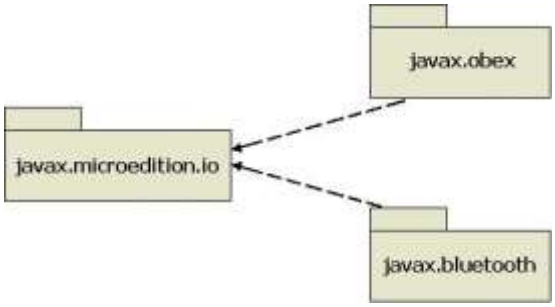


Figura 3.2 Estructura de los paquetes

3.1.6.2 Centro de Control Bluetooth (Bluetooth Control Center -BCC)

Los dispositivos Bluetooth, sobre todo los que implementen esta API, pueden permitir que múltiples aplicaciones se ejecuten simultáneamente, esto establece la necesidad de un BCC para prevenir que una aplicación afecte de manera adversa otra aplicación. El BCC es un conjunto de capacidades que le permiten a un usuario definir valores específicos para ciertos parámetros de configuración en la pila de protocolos Bluetooth y resolver peticiones de solución a conflictos hechas por las mismas aplicaciones. El BCC es la autoridad central en la configuración local de dispositivos Bluetooth, puede ser una aplicación local, una aplicación con una API separada o simplemente un grupo de datos que son especificados por el fabricante y no pueden ser cambiados por el usuario.

3.1.6.2.1 BCC y el Modo de Seguridad

En el nivel más básico, el BCC define la configuración de seguridad para el dispositivo. Por ejemplo, el BCC controla el modo de seguridad que una pila de protocolos utiliza y mantiene la lista de dispositivos confiables. El BCC no es una clase o una interfaz especifica en la API, pero es una parte importante para la seguridad de la arquitectura en esta especificación. Las API's de Java para la tecnología inalámbrica Bluetooth requieren de la existencia de un BCC que puede o no desarrollarse usando Java como lenguaje de programación.

3.1.6.3 Modelo Cliente/Servidor

Un servicio Bluetooth es una aplicación que actúa como un servidor el cual proporciona algún tipo de asistencia a dispositivos clientes a través de comunicaciones Bluetooth; generalmente esta asistencia es una capacidad o función que no está disponible localmente en el dispositivo cliente. Teniendo en cuenta los perfiles Bluetooth se pueden encontrar ejemplos de Servidores de Aplicaciones Bluetooth: Servidores de Acceso a una LAN, Servidores de Objetos y Archivos, Sincronización de Servicios, entre otros. Se pueden definir Servidores de Aplicaciones Bluetooth

además de los especificados en los perfiles y hacer que éstos se encuentren disponibles a clientes remotos definiendo un registro de servicio que describa el servicio y agregando este a la base de datos de descubrimiento del servicio (SDDB) del dispositivo local.

Una vez el registro de servicio ha sido almacenado en la SDDB, la aplicación servidor espera por una aplicación cliente para iniciar el contacto con el servidor de acceso al servicio, de esta manera la aplicación cliente y la aplicación servidor establecen una conexión Bluetooth para negociar sus requerimientos.

A continuación, se define las responsabilidades de la aplicación servidor, de la aplicación cliente, y de la pila Bluetooth.

Las responsabilidades básicas de un Servidor de Aplicaciones Bluetooth son:

- Crear un registro de servicio que describa el servicio ofrecido por la aplicación.
- Agregar un registro de servicio a la SDDB del servidor para que el servicio quede disponible a otros dispositivos.
- Establecer las medidas de seguridad asociadas a este servicio para fortalecer la conexión con los clientes.
- Aceptar las conexiones de los clientes que solicitan el servicio ofrecido por la aplicación.
- Actualizar el registro del servicio en la SDDB del servidor cada vez que las características del servicio cambien.
- Borrar o desactivar el registro del servicio de la SDDB del servidor cuando el servicio no ha estado disponible por algún tiempo.

Las responsabilidades básicas de una Aplicación Cliente Bluetooth son:

- Usar el SDP para indagar a una SDDB remota acerca de algún servicio deseado.
- Establecer las medidas de seguridad asociadas a este servicio para fortalecer la conexión con el servidor.
- Iniciar conexiones a servidores que ofrecen los servicios deseados.
- Opcionalmente, revisar la SDDB para determinar si el servicio ha cambiado o no esta disponible.

Las responsabilidades de la pila de protocolos de Bluetooth con aplicaciones Servidor Bluetooth son:

- Proporcionar un lugar para los registros del servicio que permiten a los servidores adicionar, actualizar y eliminar sus propios registros de servicio.
- Permitir el establecimiento de conexiones lógicas con las aplicaciones del cliente.

Las responsabilidades de la pila de protocolos de Bluetooth con aplicaciones Cliente Bluetooth son:

- Permitir la búsqueda y recuperación de registros de servicios almacenados en la SDDB del servidor.
- Permitir el establecimiento de conexiones lógicas con las aplicaciones del servidor.

La movilidad en los dispositivos inalámbricos implica la necesidad de encontrar una forma para conectarse a otros dispositivos y aprender acerca de lo que dichos dispositivos pueden llegar a hacer. Esta especificación proporciona API's que permiten descubrir dispositivos, encontrar servicios y ofrecer servicios a otros dispositivos. En las secciones siguientes se explica esto.

3.1.7 Descubrimiento del dispositivo

Una aplicación puede obtener una lista de los dispositivos que están dentro de su rango de cobertura usando los métodos `startInquiry()`o `retrieveDevices()`, los cuales están dentro de la clase DiscoveryAgent del paquete Javax.Bluetooth. Para utilizar el método `startInquiry()` se requiere que la aplicación especifique una clase que permanentemente escucheo (listener); entonces este listener notifica cuando se encuentran nuevos dispositivos a través de una indagación. Si una aplicación no desea esperar por la respuesta a una indagación, la API proporciona el método `retrieveDevices()` el cual retorna la lista de dispositivos que ya se habían encontrado en una indagación anterior, es decir, dispositivos clasificados como ya conocidos, este método no realiza una indagación, pero proporciona una manera rápida de conseguir una lista de dispositivos que puedan estar dentro del área. Una vez que se descubre un dispositivo, se inicia una búsqueda de servicio.

3.1.7.1 Clases para el Descubrimiento del dispositivo

Para el descubrimiento del dispositivo la especificación proporciona una interfaz y una clase:: *DiscoveryListener* y *DiscoveryAgent*, respectivamente, las cuales se encuentran dentro del paquete *javax.bluetooth*.

Interfaz javax.bluetooth.DiscoveryListener

Esta interfaz le permite a una aplicación especificar un listener que responderá a los eventos relacionados con las indagaciones que se hayan hecho. Dentro de esta interfaz se encuentra el

método `deviceDiscovered()` que es llamado cada vez que un dispositivo se encuentra durante una indagación. Cuando la indagación se completa o se cancela, el método `inquiryCompleted()`, que esta dentro de esta misma interfaz, se llama; este método recibe como argumento cualquiera de las siguientes constantes INQUIRY_COMPLETED, INQUIRY_ERROR o INQUIRY_TERMINATED para diferenciar entre una indagación que se completo, produjo un error o fue cancelada, respectivamente.

Clase javax.bluetooth.DiscoveryAgent

Esta clase proporciona métodos para el descubrimiento tanto del dispositivo como del servicio. Para el descubrimiento del dispositivo, implementa el método `startInquiry()` para fijar el dispositivo local en el modo de indagación y el método `retrieveDevices()` para que se retorne información sobre dispositivos que se encontraron en una indagación previamente realizada por el dispositivo local. También proporciona una manera de cancelar una indagación a través del método `cancelInquiry()`.

3.1.8 Descubrimiento del servicio

Como ya se mencionó un cliente puede descubrir algunos servicios que se encuentran disponibles sobre un servidor de descubrimiento del servicio. La clase *DiscoveryAgent* proporciona los métodos para buscar los servicios sobre un dispositivo servidor Bluetooth e iniciar las transacciones relacionadas con el descubrimiento del dispositivo y del servicio.

El proceso por el cual un cliente puede descubrir los servicios esta descrito en el Perfil de Aplicación de Descubrimiento del Servicio (SDAP – Sección 2.4). Esta especificación soporta las siguientes funcionalidades SDAP:

- Buscar servicios de una clase de servicio en particular.
- Recuperar los atributos del servicio.
- Simultáneamente buscar servicios y recuperar sus atributos.
- Terminar una transacción de búsqueda del servicio.

Para descubrir los servicios que se encuentran disponibles en un servidor de descubrimiento del servicio, la aplicación del cliente primero recupera un objeto que encapsula la funcionalidad SDAP, dicho objeto además de ser de tipo *DiscoveryAgent* es un objeto global. El código que permite recuperar el *DiscoveryAgent* se muestra a continuación:

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

3.1.8.1 Clases para el Descubrimiento del Servicio

El Descubrimiento del Servicio involucra cinco clases: *UUID*, *DataElement*, *ServiceRecord*, *DiscoveryAgent* y *DiscoveryListener*; las cuales se encuentran incluidas dentro del paquete *javax.bluetooth*.

Clase javax.bluetooth.UUID

Esta clase se utiliza como representación de un identificador único universal el cual se emplea como valor para un atributo de servicio, para ello se encapsula un entero sin signo de 16, 32 o 128 bits. La especificación Bluetooth define algunos UUIDs "cortos" de 16 o 32 bits y además describe cómo se puede convertir un UUID "corto" en un UUID de 128 bits, esto llega a ser necesario a la hora de comparar UUIDs de 128 bits.

Clase javax.bluetooth.DataElement

Dentro de esta clase se definen los tipos de datos que un atributo del servicio Bluetooth puede asumir. Los tipos de datos válidos son:

- Enteros con y sin signo de 1, 2, 4, 8 o 16 Bytes de longitud.
- Cadenas (String).
- Boléanos (Boolean)
- UUID
- Una secuencia de cualquiera de los anteriores tipos de datos.

Además, con esta clase se tiene una interfaz que permite construir y recuperar el valor de un atributo del servicio.

Interfaz javax.bluetooth.ServiceRecord

Dentro de esta interfaz se define el Registro del Servicio Bluetooth, el cual contiene el ID[19] y el valor del atributo. El ID del atributo es un entero sin signo de 16 bits mientras que el valor del atributo se refiere a un *DataElement*[20]. Esta interfaz además de proporcionar el dispositivo servidor remoto desde el cual se obtuvo un *ServiceRecord* define el método `populateRecord()` con el cual se recuperan los atributos del servicio deseado.

Clase javax.bluetooth.DiscoveryAgent

La clase *DiscoveryAgent* proporciona los métodos que permiten realizar el descubrimiento tanto de dispositivos como de servicios. Además, ofrece una manera de cancelar una transacción de búsqueda del servicio.

[19] ID hace referencia a un numero que generalmente se emplea como identificador.
[20] Un *DataElement* es un valor que permite describir el tipo de dato.

Interfaz javax.bluetooth.DiscoveryListener

Esta interfaz le permite a una aplicación especificar un listener que responderá a los eventos relacionados con el descubrimiento dispositivos y del servicio. Cada vez que se descubra un servicio se llama al método `servicesDiscovered()`, mientras que el método `serviceSearchCompleted()` se invoca cuando una transacción de búsqueda del servicio se completa o se cancela.

3.1.9 Registro del Servicio

Como se estudió en la sección 3.1.6.3 dentro de las responsabilidades básicas de un Servidor de Aplicaciones Bluetooth están:

1. Crear un registro de servicio que describa el servicio ofrecido por la aplicación.
2. Agregar un registro de servicio a la SDDB del servidor para que el servicio quede disponible a otros dispositivos.
3. Establecer las medidas de seguridad asociadas a este servicio para fortalecer la conexión con los clientes.
4. Aceptar las conexiones de los clientes que solicitan el servicio ofrecido por la aplicación.
5. Actualizar el registro del servicio en la SDDB del servidor cada vez que las características del servicio cambien.
6. Borrar o desactivar el registro del servicio de la SDDB del servidor cuando el servicio no ha estado disponible por algún tiempo.

Las responsabilidades 1, 2, 5, y 6 comprenden un subconjunto de las responsabilidades que el servidor tiene que hacer para anunciar un servicio a los dispositivos cliente. A este subconjunto se le conoce como Registro del Servicio.

3.1.9.1 Responsabilidades del Registro del Servicio

La sección anterior describe el registro del servicio desde una perspectiva de Bluetooth muy general. Para esta especificación, las responsabilidades del registro del servicio resultan de un esfuerzo colaborativo entre la aplicación del servidor, la implementación de la API y la pila de protocolos de Bluetooth. La Figura 3.3 describe detalladamente este aspecto.

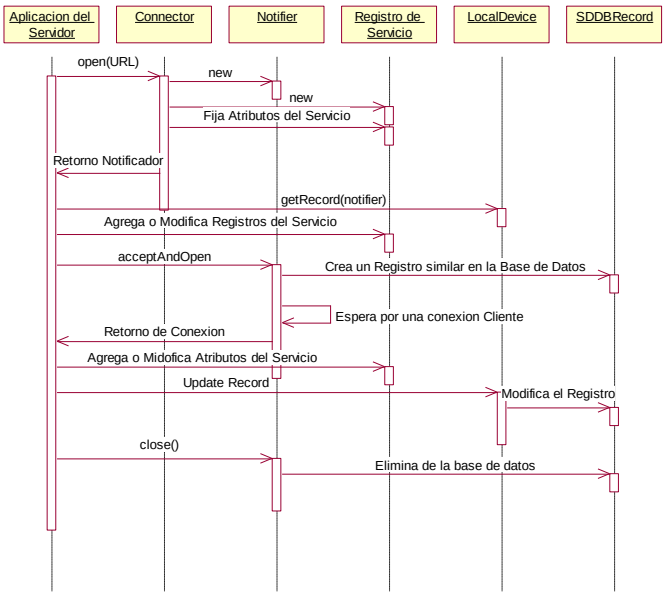


Figura 3.3 Colaboración entre la implementación de la API y la aplicación servidor para el Registro del Servicio

Como se observa en la Figura 3.3, cuando la aplicación servidor ejecuta el método *`Connector.open()`*[21] el cual tiene como argumento una cadena de Conexión URL para un servidor, entonces la aplicación crea un nuevo *ServiceRecord*; además cuando la aplicación llama al método `acceptAndOpen()`, el cual pertenece a la clase *DiscoveryListener*, adiciona un correspondiente registro de servicio a la SDDB. Luego, la aplicación del servidor puede acceder a su propio *ServiceRecord* invocando al método `getRecord()` y entonces realizar las modificaciones correspondientes para ese *ServiceRecord*, debe ser claro que estas modificaciones también se ven reflejadas en el correspondiente registro de servicio en la SDDB llamando al método `updateRecord()`. Finalmente, el registro de servicio de la aplicación se puede eliminar de la SDDB cuando la aplicación servidor envía al notificador un mensaje de cerrar el servicio.

3.1.9.2 Servicios Connect-Anytime

En el caso de la Figura 3.3 se asume que la aplicación servidor ya esta corriendo y además debe estar preparada para aceptar conexiones con un cliente; las aplicaciones servidor que deben cumplir con este requerimiento se denominan servicios *"run-before-connect"*. Algunos dispositivos tienen la capacidad de iniciar aplicaciones en el servidor sobre demanda, es decir, esto ocurre

[21] Connector.open() es un método de la clase Connector que se encuentra en el paquete javax.microedition.io permite especificar el tipo de comunicación ya sea http, Sockets, Datagramas, etc.

cuando una aplicación cliente intenta conectarse a una aplicación servidor que no se esta ejecutando; las aplicaciones servidor con esta capacidad son llamadas servicios *"connect-anytime"*.
En el caso de los servicios *"connect-anytime"*, el registro de servicio debe permanecer en la SDDB aunque la aplicación del servidor finalice debido a que un cliente puede permanecer conectado a este servicio. Idealmente, un registro de servicio debe ser descubierto por los clientes siempre que para los clientes sea posible conectarse a dicho servicio, aunque no en todos los casos es fácil lograr este objetivo, resulta muy útil para establecer las políticas que permitan adicionar y eliminar los registros de servicio de la SDDB.

Para el caso de los servicios *"run-before-connect"*, los clientes no tienen ninguna posibilidad de conectarse hasta que el servidor llame al método `acceptAndOpen()`, por consiguiente la aplicación no debe adicionar un registro de servicio a la SDDB hasta que el método `acceptAndOpen()` sea llamado. Una vez el notificador se cierra, no es posible llamar al método `acceptAndOpen()` para aceptar otra conexión del cliente, así pues, la aplicación debe eliminar el registro de servicio de la SDDB o debe desactivarlo.

En los servicios *"connect-anytime"*, la aplicación debe agregar el registro de servicio a la SDDB cuando el dispositivo y la aplicación servidor alcancen un estado dónde los clientes puedan conectarse. Cuando una aplicación servidor *"connect-anytime"* esta corriendo y ha llamado al método `acceptAndOpen()`, debe estar en este estado y tener su registro de servicio en la SDDB. Sin embargo, el registro de servicio puede agregarse a la SDDB antes, siempre y cuando los clientes pueden conectarse en este punto. En algunos casos, los clientes pueden conectarse en cuanto una aplicación servidor se instala en un dispositivo. En estos casos, un registro de servicio puede agregarse a la SDDB en el momento de la instalación de la aplicación.

El registro del servicio debe removerse de la SDDB o deshabilitarse cuando las aplicaciones cliente ya no puedan conectarse al servicio. Por ejemplo, el registro de servicio para un servicio *"connect-anytime"* debe removerse o deshabilitarse durante el momento de instalación de la aplicación debido a que los clientes no se pueden conectar a dicho servicio en este dispositivo.
Para las aplicaciones que se basan en esta especificación no es necesario soportar ambos servicios *"run-before-connect"* y *"connect-anytime"*, basta con soportar uno de los dos tipos de servicios.

3.1.9.3 Modos Conectable y No-Conectable

En la sección 2.2.9.2 se describieron los modos de operación que caracterizan a los dispositivos Bluetooth:

- **Modo Conectable:** En este estado el dispositivo escucha periódicamente los intentos de otro dispositivo remoto para iniciar una conexión.
- **Modo No Conectable:** Un dispositivo en este modo no escucha los intentos de otro dispositivo remoto para iniciar una conexión.

Las siguientes funciones del cliente sólo tendrán éxito si el dispositivo servidor está en el modo Conectable:

- Usar el SDP para indagar a una SDDB remota por los servicios deseados.
- Iniciar conexiones a servidores que ofrecen los servicios deseados.
- Opcionalmente, revisar la SDDB remota para determinar si el servicio ha cambiado o no esta disponible.

Para el funcionamiento adecuado de una aplicación servidor es necesario que el dispositivo servidor sea conectable, por lo tanto, en esta especificación se trata de hacer que los dispositivo locales sean conectables siempre que la aplicación detecte la existencia de registros de servicios en la SDDB del dispositivo local. Como parte de la implementación del método `acceptAndOpen()`, se debe garantizar que el dispositivo local sea conectable. Para el caso de servicios *"connect-anytime"* otra instancia del método `acceptAndOpen()` puede originar que la aplicación verifique la existencia de registros de servicios y solicite al dispositivo servidor entrar en el modo conectable, sin embargo, esto depende estrictamente de la aplicación.

Como las razones de si un dispositivo es o no conectable dependen directamente de los usuarios, la aplicación no tiene la decisión final para definir si el dispositivo entrará o no en el modo conectable. Para que el dispositivo local sea conectable la aplicación le hace una solicitud al BCC, pero dicha solicitud podría no ser rechazada siempre y cuando el usuario del dispositivo haya elegido establecer el dispositivo local en el modo No Conectable. Cuando el dispositivo servidor intenta poner a dicho dispositivo local en modo conectable, una *BluetoothStateException* es lanzada, debido a que la solicitud entra en conflicto con la configuración del dispositivo establecida por el usuario.

Cuando todos los registros de Servicio en la SDDB han sido eliminados o deshabilitados, la aplicación opcionalmente puede solicitar que el dispositivo servidor entre en el modo No Conectable.

Aunque un dispositivo en el modo No Conectable no responde a las peticiones de conexión hechas por dispositivos remotos, si podría iniciar peticiones de conexión propias. Es decir, un dispositivo

en el modo No Conectable puede ser un cliente, pero no un servidor. Por esta razón, la aplicación no necesita la solicitud de modo Conectable para un dispositivo sin ningún Registro de Servicio en su SDDB.

3.1.9.4 Clases para el Registro del Servicio

El Registro del Servicio involucra una interfaz denominada: *ServiceRecord* y las siguientes clases *LocalDevice* y *ServiceRegistrationException*; las cuales se encuentran incluidas dentro del paquete *javax.bluetooth*.

Interfaz javax.bluetooth.ServiceRecord

Un registro de servicio describe un servicio Bluetooth a los clientes. Los Registros de Servicio están compuestos por un conjunto de atributos del servicio, donde cada atributo es una pareja que consiste de un ID del atributo y un valor del atributo.

Un servidor SDP ofrecido por la pila de protocolos Bluetooth mantiene una "Base de Datos" de registros de servicio que describen los servicios en el dispositivo servidor. En el caso de un servicio *"run-before-connect"* adiciona su *ServiceRecord* a la SDDB llamando al método `acceptAndOpen()`, de esta forma los clientes del Descubrimiento del servicio usan el SDP para consultar al servidor SDP por los registros de servicio de interés, como se puede ver en la Figura 3.4; finalmente un *ServiceRecord* brinda la información suficiente para que un cliente SDP pueda conectarse al servicio en el dispositivo servidor.

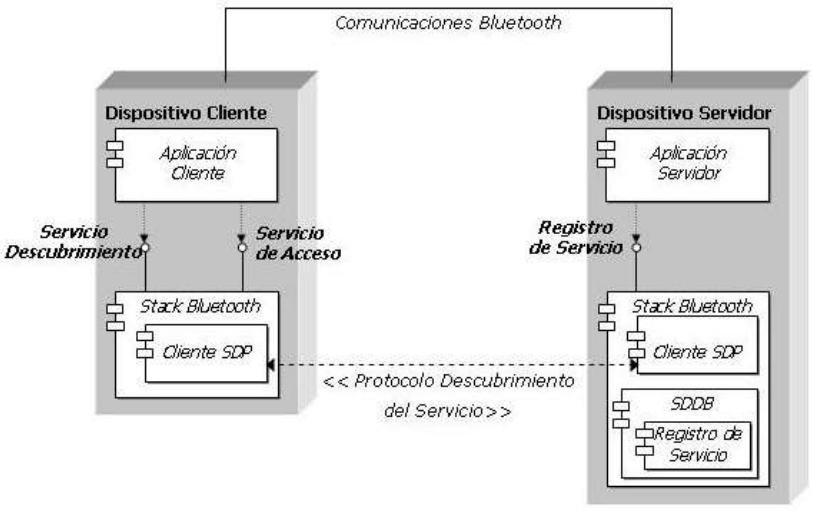


Figura 3.4 Un servidor proporciona un Registro de Servicio que habilita la conexión con el cliente.

Clase javax.bluetooth.LocalDevice

La clase *LocalDevice* proporciona un método `getRecord()` el cual puede utilizar una aplicación servidor para obtener su *ServiceRecord*, luego el servidor puede modificar el objeto *ServiceRecord* adicionando o modificando sus atributos, una vez terminado esto, el registro de servicio actualizado puede almacenarse en la SDDB realizando un *`notifier.acceptAndOpen()`* o usando el método *`updateRecord()`* de *LocalDevice*.

Clase javax.bluetooth.ServiceRegistrationException

Cuando un intento por adicionar o modificar un registro de servicio falla, se lanza la excepción *ServiceRegistrationException*, dichas fallas en el Registro del Servicio pueden ocurrir:

- Durante la ejecución del método *`Connector.open()`*, ya que la aplicación crea un nuevo registro de servicio para el servicio especificado por el método `Connector.open()`.
- Cuando un servicio *"run-before-connect"* invoca el método *acceptAndOpen()* y la aplicación intenta adicionar el registro de servicio asociado con el notificador de la SDDB.
- Después de la creación inicial del registro de servicio, cuando la aplicación servidor intenta modificar el registro de servicio en la SDDB usando el método *`updateRecord()`.*

Es posible cambiar la manera de cómo el dispositivo local responde a los dispositivos remotos: una es utilizando las clases que representan los objetos Bluetooth esenciales como *LocalDevice* y *RemoteDevice*, haciendo uso de los métodos que acceden a las propiedades de estos objetos, como sus nombres y direcciones Bluetooth y/o llamando a los métodos que manejan los estados del LocalDevice, es decir, como hacer el dispositivo encontrable.

3.1.10 Perfil de Acceso Genérico GAP

Existen clases que representan los objetos de Bluetooth esenciales tales como *LocalDevice* y *RemoteDevice*, estas dos clases proporcionan las capacidades de gestión del dispositivo que forman parte del GAP. Los métodos de control estándar para el dispositivo local están en la Clase *LocalDevice*, el soporte a ésta clase lo brindan las clases *DeviceClass* y *BluetoothStateException*. DeviceClass tiene métodos que permiten recuperar los valores para las clases de servicio y para las clases de dispositivos que describen las propiedades de un dispositivo. Finalmente, la *clase RemoteDevice* representa un dispositivo remoto y proporciona los métodos para recuperar información acerca de dicho dispositivo remoto.

3.1.10.1 Clases para el Perfil de Acceso Genérico

Para el Perfil de Acceso Genérico se han definido cuatro clases: *LocalDevice*, *RemoteDevice*, *BluetoothStateException* y *DeviceClass*; las cuales se encuentran incluidas dentro del paquete *javax.bluetooth*.

Clase javax.bluetooth.LocalDevice

Esta clase permite el acceso y el control del dispositivo Bluetooth local, además esta diseñada para cumplir los requerimientos del GAP como esta definido en la especificación Bluetooth.

Clase javax.bluetooth.RemoteDevice

Esta clase representa un dispositivo Bluetooth remoto, proporciona información básica sobre un dispositivo remoto, inclusive las direcciones de los dispositivos Bluetooth y su nombre (nombre del dispositivo Bluetooth).

Clase javax.bluetooth.BluetoothStateException

Esta excepción se ejecuta cuando un dispositivo no puede atender alguna petición que normalmente soporta. Por ejemplo, algunos dispositivos no permiten alguna indagación cuando dicho dispositivo esta conectado a otro dispositivo.

Clase javax.bluetooth.DeviceClass

Esta clase define los valores tanto para el tipo de dispositivo como para los tipos de servicios en un dispositivo.

3.1.11 Seguridad

Los dispositivos inalámbricos son potencialmente más vulnerables a falsificaciones del mensaje original que los dispositivos alambricos, Bluetooth incluye varias respuestas a esta vulnerabilidad, y además de algunas capacidades como salto de frecuencia existen otras capacidades, como la encriptación y la autenticación, que pueden ser o no utilizadas de acuerdo a las necesidades de las aplicaciones.

Las aplicaciones Cliente y servidor pueden agregar opcionalmente parámetros al argumento de la cadena de conexión del método `Connector.open()` con el propósito de especificar la seguridad requerida en las conexiones, esto es posible para diferentes conexiones que involucran diferentes servicios y requieren de diferentes niveles de seguridad.

Los parámetros establecidos en la cadena de conexión se pueden usar para fijar medidas de seguridad al momento de establecer la conexión. La aplicación cliente y la aplicación servidor

pueden usar cuando lo deseen los métodos de la clase *RemoteDevice* para solicitar un cambio en la seguridad de una conexión en particular.

3.1.11.1 Requerimientos de seguridad en la Cadena de Conexión

Las aplicaciones servidor usan uno de los métodos *open* de la clase *javax.microedition.io.Connector* del CLDC para crear un objeto notificador el cual se puede usar con el propósito de esperar por la conexión del cliente. Para un servidor, los componentes obligatorios de los argumentos en la cadena de conexión del método *open* proporcionan la suficiente información con el fin de crear un objeto de la clase apropiada del notificador y además para crear el registro de servicio apropiado. Sin embargo, se pueden agregar algunos parámetros opcionales a la cadena de conexión con el propósito de especificar los requerimientos del servidor para las conexiones con los clientes, generalmente estos parámetros opcionales se usan para autenticación, encriptación, autorización y cambio de rol maestro/esclavo.

3.1.11.2 Requerimientos del servidor para la Autenticación

La autenticación Bluetooth, es un medio que permite verificar la identificación de un dispositivo remoto. La autenticación involucra un intercambio entre dispositivos y un esquema de respuesta que requiere 128 bits como clave de enlace compartida, la cual se deriva de un código PIN compartido por ambos dispositivos, si el código PIN en ambos dispositivos no es igual, el proceso de autenticación falla.

El parámetro *authenticate* tiene la siguiente interpretación cuando se usa en la cadena de conexión de una aplicación servidor:

- Si *authenticate=true*, la aplicación trata de verificar la identidad de cada dispositivo cliente que intenta conectarse al servicio.
- Si *authenticate=false*, la aplicación no trata de verificar la identidad de cada dispositivo cliente que intenta conectarse al servicio.
- Si el parámetro *authenticate* no está presente en la cadena de conexión, entonces la aplicación no intenta verificar la identidad de los clientes, a menos que otros parámetros presentes en la cadena de conexión requieran este chequeo de identidad.

No todos los sistemas Bluetooth soportan la autenticación, algunas veces aunque se soporte la autenticación, es posible que aunque el parámetro *authenticate=trae,* exista un conflicto con la configuración de seguridad del dispositivo que el usuario ha establecido a través del BCC. Una *BluetoothConnectionException* se ejecuta en el método `Connector.open()` si *authenticate=true* y no se soporta la autenticación, o no soporta los conflictos de autenticación con la configuración

de seguridad del dispositivo. Si hay un conflicto entre las necesidades de seguridad de una aplicación y la configuración de seguridad del dispositivo, algunas aplicaciones BCC pueden intentar eliminar el conflicto solicitándole al usuario que considere cambiar la configuración del dispositivo.

3.1.11.3 Requerimientos del servidor para Encriptación

La encriptación se puede aplicar a las comunicaciones basadas en enlaces de datos entre dos dispositivos Bluetooth. Cuando se activa, la encriptación se aplica a todos los datos transferidos en ambas direcciones del enlace.

El parámetro *encrypt* tiene la siguiente interpretación cuando se usa en la cadena de conexión de una aplicación servidor:

- Si *encrypt=true*, la aplicación encripta toda la comunicación del servicio.
- Si *encrypt=false*, la aplicación servidor no requiere encriptación, pero se puede usar si es necesaria en el dispositivo cliente o por otras conexiones existentes sobre el enlace de datos entre los dos dispositivos.
- Si el parámetro *encrypt* no está presente en la cadena de conexión, esto es equivalente a *encrypt=false*.

Debido a que la encriptación Bluetooth requiere una clave de enlace compartida, esto conduce a que la encriptación requiere autenticación, lo que significa que sólo ciertas combinaciones de configuración de parámetros son válidas:

- *authenticate=true* y *encrypt=true* es una combinación válida.
- *authenticate=true* y *encrypt=false* es una combinación válida.
- *authenticate=false* y *encrypt=false* es una combinación válida.
- *authenticate=false* y *encrypt=true* no es una combinación válida, además genera una *BluetoothConnectionException*.
- *encrypt=true* con el parámetro *authenticate* ausente, se trata como el equivalente a *authenticate=true*.

Al igual que en la autenticación, no todos los sistemas Bluetooth soportan la encriptación. En algunos casos aunque se soporte la encriptación, es posible que además del parámetro *encrypt=true* exista un conflicto con la configuración de seguridad del dispositivo que el usuario ha establecido a través del BCC. Una *BluetoothConnectionException* se ejecuta en el método `Connector.open()` si *encrypt=true* y no se soporta la encriptación o la misma entra en conflicto con la configuración de seguridad del dispositivo.

3.1.11.4 Requerimientos del servidor para la Autorización

La autorización Bluetooth es un procedimiento en el cual un usuario del dispositivo servidor le permite el acceso a un servicio específico a un dispositivo cliente específico. La aplicación de autorización puede preguntarle al usuario del dispositivo servidor si el dispositivo cliente tiene permiso para acceder al servicio, también puede consultar una lista de dispositivos que son "confiables" y por consiguiente permitirle acceder a todos los servicios. El parámetro *authorize* tiene la siguiente interpretación cuando se utiliza en la cadena de conexión de una aplicación servidor:

- Si *authorize=true*, la aplicación consulta con el BCC para determinar si al dispositivo cliente que requiere una conexión se le debe permitir el acceso al servicio.
- Si *authorize=false*, a todos los clientes se le permite el acceso al servicio.
- Si el parámetro *authorize* no está presente en la cadena de conexión, esto es equivalente a *authorize=false*.

Al igual que la encriptación, la autorización implica que se pueda verificar la identidad del dispositivo cliente a través de la autenticación, esto significa que sólo ciertas combinaciones de configuraciones del parámetro son válidas:

- *authenticate=true* y *authorize=true* es una combinación válida.
- *authenticate=true* y *authorize=false* es una combinación válida.
- *authenticate=false* y *authorize=false* es una combinación válida.
- *authenticate=false* y *authorize=true* no es una combinación válida, que genera una *BluetoothConnectionException*.
- *authorize=true* con el parámetro *authenticate* ausente se trata como el equivalente a *authenticate=true*.

Como en el caso de la autenticación y encriptación, no todos los sistemas Bluetooth soportan la autorización. Y aunque la autorización se soporte, es posible que además del parámetro *authorize=true* exista un conflicto con la configuración de seguridad del dispositivo que el usuario ha establecido a través del BCC. Una *BluetoothConnectionException* se ejecuta en el método *Connector.open()* si *authorize=true* y no se soporta la autorización o la autorización entra en conflicto con la configuración de seguridad del dispositivo.

3.1.11.5 Requerimientos del servidor para el rol de Maestro

Los dispositivos Bluetooth forman una red, cada red Bluetooth tiene un dispositivo Maestro cuyo reloj y secuencia de saltos de frecuencia se usan para sincronizar hasta siete dispositivos esclavos. Un dispositivo Bluetooth puede tener el rol de Maestro o de Esclavo. El dispositivo que comienza la

formación de un enlace de datos a otro dispositivo es el que hace las veces de Maestro de la red, sin embargo, Bluetooth mantiene un procedimiento para que un dispositivo solicite cambiar su rol de maestro/esclavo y viceversa.

El parámetro *master* tiene la siguiente interpretación cuando se usa en la cadena de conexión de una aplicación servidor:

- Si *master=true*, entonces en cuanto se establezca la conexión, la aplicación solicita que el cliente y el servidor intercambien los papeles para que el servidor sea el maestro de la red.
- Si *master=false*, el servidor está dispuesto a ser el maestro o el esclavo.
- Si el parámetro *master* no está presente en la cadena de conexión, esto es equivalente a *master=false*.

No todos los sistemas Bluetooth soportan un cambio de rol maestro/esclavo. Si *master=true* y el dispositivo servidor no soporta el intercambio de rol master/esclavo, una *BluetoothConnectionException* se ejecuta en el método `Connector.open()`.

3.1.11.6 Requerimientos del cliente en la cadena de Conexión

Las aplicaciones cliente también pueden usar los parámetros *authenticate*, *encrypt* y *master* en el argumento de la cadena de conexión `Connector.open()`. Cuando se usan por los clientes, estos parámetros de conexión tienen las siguientes interpretaciones:

- Cuando *authenticate=true*, la aplicación trata de verificar la identidad del dispositivo servidor.
- Cuando *encrypt=true*, la aplicación encripta todas las comunicaciones de este servicio. Como con los servidores, *encrypt=true* implica *authenticate=true*.
- Cuando *master=true*, el cliente debe tomar el rol de maestro en las comunicaciones con el servidor, de manera que la aplicación debe negarse a que el servidor intente iniciar un intercambio del rol.

Para esta especificación, el único dispositivo que necesita conceder permisos para usar un servicio, es el dispositivo que ofrece dicho servicio; por consiguiente, el parámetro *authorize* no se permite en las conexiones del cliente, es decir, una *BluetoothConnectionException* se ejecuta si aparece *authorize=true* o *authorize=false* en una cadena de conexión cliente. Cuando un cliente intenta conectarse a un servicio ofrecido por un servidor, ambos dispositivos tienen sus propias configuraciones para los parámetros de la cadena de conexión, las configuraciones indican los requerimientos que tiene cada dispositivo para esta conexión. Casi todas las posibles combinaciones de parámetros en la cadena de conexión cliente y servidor pueden llevar a una

conexión exitosa, la única excepción es cuando el cliente y el servidor establecen *master=true*; en este caso, el intento de conexión falla debido a la disputa sobre cual dispositivo jugará el rol de maestro, el cliente es consciente de esta falla en el establecimiento de una conexión porque la llamada del cliente a `Connector.open()` ejecuta una *BluetoothConnectionException*, mientras que el servidor no es consciente del fracaso debido a que la aplicación en el lado del servidor rechaza el intento de conexión pero no ejecuta ninguna excepción, mientras tanto la aplicación servidor continúa esperando una llamada al método `acceptAndOpen()` para que la conexión sea exitosa.

3.1.11.7 Clases de seguridad

La seguridad en Bluetooth se solicita usando la clase *Connector*, del paquete *javax.microedition.io*. La clase *RemoteDevice*, del paquete *javax.bluetooth*, también tiene métodos relacionados con la seguridad.

Clase javax.bluetooth.RemoteDevice

RemoteDevice contiene métodos que se pueden usar cuando se desee solicitar un cambio en la seguridad de una conexión o indagar las configuraciones de seguridad sobre la misma conexión. En situaciones dónde se requiere un aumento en el nivel de la seguridad para un limitado conjunto de operaciones o por un corto período de tiempo, se invocan métodos que cambian las configuraciones de seguridad, algunos de estos métodos toman una instancia de *javax.microedition.io.Connection* como argumento; este tipo de argumento se usa en estos métodos para que se puedan aplicar a las conexiones de puerto serial, L2CAP y OBEX.

Para usar un servicio en un dispositivo remoto, el dispositivo local Bluetooth se puede comunicar usando los protocolos como si fueran un servicio remoto. Las aplicaciones pueden acceder adecuadamente a una amplia variedad de servicios Bluetooth a través de conexiones a servicios que tienen como protocolos de alto nivel a RFCOMM, L2CAP u OBEX; para servicios que utilizan algún otro nivel protocolar que se encuentre sobre estos tres (por ejemplo, TCP/IP), debe ser posible para una aplicación acceder a dicho servicio implementando el protocolo adicional dentro del la aplicación. La API para el Perfil del Puerto Serial proporciona una interfaz de alto nivel a muchos servicios que usan el protocolo RFCOMM.

3.1.12 Perfil de Puerto Serial

Como ya se menciono el protocolo RFCOMM ofrece la posibilidad de emular múltiples puertos seriales RS-232 entre dos dispositivos Bluetooth. Las direcciones Bluetooth de los dos dispositivos

identifican una sesión RFCOMM, solo una sesión RFCOMM puede existir entre cualquier par de dispositivos en un momento determinado, pero una sesión puede tener más de una conexión; el número de conexiones que se pueden hacer simultáneamente en un dispositivo de Bluetooth depende específicamente de la aplicación, por ello un dispositivo puede tener más de una sesión RFCOMM siempre y cuando cada sesión sea con un dispositivo diferente.

Una aplicación que ofrece un servicio basándose en el Perfil del Puerto Serial (SPP) se conoce como un servidor SPP, mientras que la aplicación que inicia una solicitud de conexión a un servicio SPP se conoce como cliente SPP; las aplicaciones cliente y servidor pueden residir en cualquier dispositivo durante una sesión RFCOMM. Inicialmente el servidor SPP registra su servicio en la SDDB, como parte del proceso de registro del servicio la aplicación adiciona al registro del servicio un *identificador de canal servidor*, para localizar el servicio, el cliente utiliza la especificación relacionada con el descubrimiento del servicio y entonces puede conectarse al servicio especificando tanto la dirección del servidor como el identificador de canal servidor. Una vez se establece la conexión, los datos se transmiten en ambas direcciones entre el cliente y servidor. La negociación de los parámetros de conexión y control de flujo entre dos dispositivos Bluetooth debe ser manejada por la aplicación de la conexión SPP.

3.1.12.1 Registro del Servicio de Puerto serial

Un servidor SPP debe inicializar los servicios que ofrece y además registrar dichos servicios en la SDDB. Un par de objetos que representan un servicio del puerto serial son:

- Un objeto que implementa la interfaz *javax.microedition.io.StreamConnectionNotifier*. Este objeto escucha todo el tiempo por las conexiones de clientes a este servicio.
- Un objeto que implementa la interfaz *javax.bluetooth.ServiceRecord*. Este objeto describe el servicio y determina cómo los dispositivos remotos pueden acceder.

Una aplicación servidor utiliza el método *Connector.open()* con una URL de conexión al Servidor SPP para crear los dos objetos que representan el servicio de puerto serial. Por ejemplo:

```
StreamConnectionNotifier service =
(StreamConnectionNotifier)Connector.open (
"btspp://localhost:102030405060708090A1B1C1D1D1E100;name=SPPEx");
```

La invocación del método *Connector.open()* con una URL de conexión al servidor SPP devuelve un *StreamConnectionNotifier* que representa el servicio SPP. La implementación del método *Connector.open()* también crea un nuevo registro de servicio que representa el servicio SPP.

Para el caso de un servicio *"run-before-connect"*, el registro de servicio se agrega a la SDDB la primera vez que la aplicación servidor llama al método *acceptAndOpen()* el cual se encuentra en el *StreamConnectionNotifier* asociado. Cuando el registro de servicio se agrega a la SDDB, entonces se hace visible a las aplicaciones cliente SPP potenciales.

3.1.12.2 Establecimiento de la conexión del servidor

Como ilustración en el siguiente código, un servidor SPP crea un objeto de tipo *StreamConnectionNotifier*:

1. Usando como argumento la cadena apropiada para un servidor SPP en el método `Connector.open()`.
2. Pasando el resultado devuelto desde `Connector.open()` a la interfaz *StreamConnectionNotifier*.

```
StreamConnectionNotifier service =
(StreamConnectionNotifier) Connector.open (
"btspp://localhost:102030405060708090A1B1C1D1D1E100;name=SPPEx");

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

El servidor emplea el método `acceptAndOpen()` mediante el cual indica que está listo para aceptar una conexión cliente, mientras tanto el método se bloquea hasta que el cliente se conecte. En el código del ejemplo anterior se puede observar que el método `acceptAndOpen()` devuelve un objeto *StreamConnection* cuando el servicio acepta una conexión requerida por un cliente. La implementación del método `acceptAndOpen()` debe permitir que la pila de protocolos de Bluetooth envíe toda la comunicación entre la aplicación cliente y la aplicación servidor a través de los flujos asociados con el objeto devuelto por el método `acceptAndOpen()`. El objeto devuelto por el método `acceptAndOpen()` debe implementar las interfaces *StreamConnection*, pero típicamente será una instancia de la clase que se adapta específicamente para el SPP.

El servicio SPP puede aceptar múltiples conexiones de diferentes clientes mediante llamadas repetidas al método `acceptAndOpen()`, para cada conexión que se acepta, se crea un nuevo objeto *StreamConnection*, sin embargo, cada cliente accede al mismo registro de servicio y se conecta al servicio usando el mismo canal servidor RFCOMM.

El método `close()` dentro del objeto *StreamConnection,* el cual representa una conexión SPP en el lado servidor, se utiliza para cerrar la conexión.

Cuando un servicio *"run-before-connect"* envía un mensaje `close()` a un *StreamConnectionNotifier*, el registro de servicio asociado con ese notificador queda inaccesible para los clientes a través del descubrimiento del servicio. La implementación debe eliminar el registro de servicio de la SDDB o usar alguna característica que ofrezca la pila de protocolos Bluetooth que permita desactivarlo, de tal forma que el registro de servicio permanezca en la SDDB pero sea inaccesible para los clientes. El mensaje `close()` también permite que la aplicación desactive algunos bits de la clase del servicio que hayan sido activados por el método `setDeviceServiceClasses()`, a menos que otro servicio cuyo notificador se este ejecutando, active los mismos bits.

Si los *StreamConnections* de este servicio permanecen abiertos cuando el *StreamConnectionNotifier* se cierra, no es posible liberar el canal servidor RFCOMM que se asigna a este servicio; sólo hasta cuando todos los *StreamConnections* de este servicio y el notificador estén cerrados la implementación puede liberar el canal servidor RFCOMM.

Si una aplicación no cierra el *StreamConnectionNotifier* o todos los *StreamConnections*, entonces la implementación del API normalmente puede realizar las operaciones de finalización cuando la aplicación termina. En el caso de un servicio *"run-before-connect"*, la aplicación debe eliminar el registro de servicio, liberar el canal servidor y desactivar los bits de la clase de servicio, a menos que otros servicios correspondientes a esos mismos bits permanezcan activos. Teniendo en cuenta la posibilidad de cierres abruptos, el registro de servicio para los servicios *"run-before-connect"* puede permanecer en la SDDB y los bits de la clase de servicio pueden permanecer activos aunque el servidor no esté corriendo.

3.1.12.3 Establecimiento de la Conexión del cliente

Antes de que un cliente SPP pueda establecer una conexión con un servicio SPP, dicho servicio debe ser descubierto mediante el descubrimiento de servicio. Una URL de conexión del cliente incluye tanto la dirección del dispositivo Bluetooth servidor como el identificador de canal del servidor para el servicio, para obtener la URL de conexión con el cliente para dicho servicio se utiliza el método `getConnectionURL()` en la interfaz *ServiceRecord*.

Una invocación al método `Connector.open()` con la URL de conexión del Cliente SPP devuelve un Objeto *StreamConnection* que representa una conexión SPP en el lado cliente. El siguiente

ejemplo muestra un cliente que establece una conexión a un servicio SPP identificado con el identificador de canal servidor = 5 en un dispositivo con dirección '0050C000321B':

```
StreamConnection con =
(StreamConnection)
  Connector.open("btspp://0050C000321B:5");
```

El método `close()` dentro del objeto *StreamConnection,* el cual representa una conexión SPP en el lado cliente, se utiliza para cerrar la conexión.

3.1.12.4 Clases de Servicio del dispositivo

Los dispositivos cliente pueden consultar la *DeviceClass* de un dispositivo servidor para obtener una idea general del tipo de dispositivo que es (por ejemplo, teléfono, PDA, o PC) y además las clases de servicio principales que se ofrecen (por ejemplo, telefonía, o información). Esto significa que hay dos maneras diferentes en que una aplicación servidor describe el servicio que ofrece:

- Agregando un registro de servicio a la SDDB.
- Activando los bits mas significativos de la clase de servicio en la *DeviceClass*.

Un servidor usa el metodo `setDeviceServiceClasses()` para asociar el *ServiceRecord* con todas las clases de servicio principales que describen dicho servicio, luego cuando hay un servicio *"run-before-connect"* primero se llama el metodo `acceptAndOpen()`, su registro de servicio y sus bits mas significativos de la clase de servicio se hacen visibles en el dispositivo cliente. En el caso de las clases de servicio principales, el metodo `acceptAndOpen()` realiza un OR del estado actual de los bits de la clase de servicio del dispositivo con las clases de servicio principales declaradas por el metodo `setDeviceServiceClasses()`. Esta operacion OR podría activar los bits adicionales de la clase de servicio que indican las nuevas capacidades para el dispositivo.
Una aplicación servidor no requiere usar el metodo `setDeviceServiceClasses()` sin embargo, es recomendable que un servidor use este método para describir su servicio en terminos de las clases de servicio principales, esta práctica le permite a los clientes obtener un *DeviceClass* para el servidor que con precisión describe las clases de servicio principales proporcionadas por el servidor.

3.1.13 Protocolo de Control y Adaptación de Enlace Lógico

L2CAP soporta dos tipos de conexiones, la primera se conoce como *orientada a la conexión* (bidireccional) y la segunda se denomina s*in conexión* (unidireccional), estas conexiones se hacen usando la primitiva de servicio *connect* del paquete *javax.microedition.io* como se puede ver en la

Figura 3.5, la cual es proporcionada por la capa L2CAP orientada a la conexión del stack de protocolos de Bluetooth. Para establecer los canales de datos sin conexión se utiliza el concepto de comunicación de grupo proporcionado por la capa L2CAP.

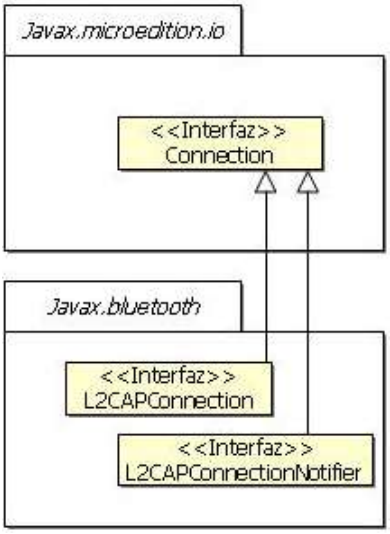


Figura 3.5 Marco de Conexión L2CAP

3.1.13.1 Configuración del canal

Los canales orientados a la conexión necesitan ser configurados una vez se establece la conexión. Los parámetros relacionados con la configuración del canal que se negocian entre los dispositivos Bluetooth son:

- Unidad de Transmisión Máxima (MTU) – Hace referencia al máximo tamaño de la carga útil (en bytes) que el remitente es capaz de enviar. La aplicación puede especificar la MTU entrante que le gustaría usar durante la conexión; si dicho valor no se especifica, entonces se emplea el *DEFAULT_MTU* de 672 bytes; así mismo la aplicación puede especificar el MTU deseado para el dispositivo remoto, es decir, el MTU saliente, si dicho valor no se especifica, entonces éste será menor o igual al MTU entrante anunciado por el dispositivo remoto durante la configuración del canal.
- Interrupción Flush – Se refiere a la cantidad de tiempo para el cual la gestión del enlace intentará transmitir un paquete con éxito antes de vaciar el paquete. Un valor de *0xFFFF*[22] indica que el paquete se transmitirá hasta que sea reconocido o hasta que el enlace ACL termine, este valor brinda una fiabilidad en el enlace de comunicación. L2CAP proporciona un canal de comunicación full-dúplex el cual entrega las unidades de datos del protocolo

[22] Para esta especificación el valor de la Interrupción Flush por defecto - 0xFFFF - se emplea para la conexión.

L2CAP de manera ordenada, sin embargo no se ofrece ningún mecanismo que permita asegurar una transmisión fiable de las unidades de datos del protocolo, no obstante cuenta con un proceso de retransmisión a nivel de banda-base que permite soportar suficientes canales de comunicación confiables para las capas más altas.

- Calidad de Servicio (QoS) – La pila de protocolos de Bluetooth determina los valores de QoS, sin embrago este patrón hace referencia a una descripción de flujo del tráfico.

3.1.13.1.1 Unidad de Transmisión máxima (MTU)

Antes de que cualquier operación de lectura/escritura pueda llevarse a cabo durante el proceso de conexión, la aplicación debe responder por la configuración del canal brindando una MTU ya sea predefinido o requerido al usuario. El *ReceiveMTU* es el número máximo de bytes que el dispositivo local puede recibir en la carga útil. El *TransmitMTU* es el número máximo de bytes que el dispositivo local puede enviar al dispositivo remoto en la carga útil. Sí *DevA* es el dispositivo local y *DevB* es el dispositivo remoto, se definen las siguientes variantes para el *ReceiveMTU*:

- $ReceiveMTU_A$: Es el máximo tamaño de la carga útil propuesto por una aplicación en el DevA para las cargas útiles L2CAP recibidas en el *DevA*.
- $ReceiveMTU_B$: Es el máximo tamaño de la carga útil propuesto por una aplicación en el DevB para las cargas útiles L2CAP recibidas en el *DevB*.
- $ReceiveMTU_{AB}$: Es el máximo tamaño de la carga útil acordado por el *DevA* y el *DevB* para cargas útiles L2CAP recibidas por una aplicación en el *DevA*.

Ocurre lo mismo para el *TransmitMTU*:

- $TransmitMTU_A$: Es el máximo tamaño de la carga útil propuesto por una aplicación en el DevA para las cargas útiles L2CAP enviadas por el *DevA.*
- $TransmitMTU_B$: Es el máximo tamaño de la carga útil propuesto por una aplicación en el DevB para las cargas útiles L2CAP enviadas por el *DevB*.
- $TransmitMTU_{AB}$: Es el máximo tamaño de la carga útil acordado por el *DevA* y el *DevB* para cargas útiles L2CAP enviadas por una aplicación en el *DevA*.
- Si $ReceiveMTU_A \geq TransmitMTU_B$, entonces $ReceiveMTU_{AB} \leq ReceiveMTU_A$. Y si $ReceiveMTU_A < TransmitMTU_B$, entonces la conexión entre *DevA* y *DevB* falla.
- Si $TransmitMTU_A \leq ReceiveMTU_B$, entonces $TransmitMTU_{AB} = TransmitMTU_A$. Y si $TransmitMTU_A > ReceiveMTU_B$, entonces la conexión entre *DevA* y *DevB* falla.

Cuando la aplicación en el dispositivo local, DevA, invoca:

`Connector.open("btl2cap://...;ReceiveMTU=1024;TransmitMTU=512"),`

$ReceiveMTU_A = 1024$ y $TransmitMTU_A = 512$.

Cuando la aplicación en el dispositivo remoto, DevB, invoca:

`Connector.open("btl2cap://...;ReceiveMTU=2048;TransmitMTU=512")`

$ReceiveMTU_B$ = 2048 y $TransmitMTU_B$ = 512. Para este caso, se puede establecer una conexión entre *DevA* y *DevB* debido a que $ReceiveMTU_{AB} \leq 1024$ y $TransmitMTU_{AB} = 512$.

Existen varias maneras de que la aplicación configure la MTU cuando se hace una solicitud de conexión:

1. La aplicación especifica el $ReceiveMTU_A$ y el $TransmitMTU_A$. En este caso, la aplicación advierte al dispositivo remoto el valor del $ReceiveMTU_A$ en la solicitud de la configuración, si el dispositivo remoto retorna una respuesta de configuración negativa, la conexión se considera no exitosa, mientras que si el dispositivo remoto retorna una respuesta de configuración positiva, la aplicación queda esperando la solicitud de configuración del dispositivo remoto. Cuando el dispositivo local recibe una solicitud de configuración desde el dispositivo remoto, compara los valores del $ReceiveMTU_B$ en la solicitud entrante al $TransmitMTU_A$ y si el tamaño máximo que la aplicación planea enviar, es decir $TransmitMTU_A$, resulta ser menor o igual al tamaño máximo que el dispositivo remoto puede recibir, es decir $ReceiveMTU_B$, la conexión tiene éxito; de otra manera la conexión falla.
2. La aplicación especifica el $ReceiveMTU_A$, sin embargo no especifica el $TransmitMTU_A$. En este caso, la configuración con respecto al $ReceiveMTU_A$ es similar al párrafo anterior, pero en el caso que en la solicitud de configuración recibida desde el dispositivo remoto el $TransmitMTU_{AB}$ sea menor o igual al $ReceiveMTU_B$ la aplicación debe usar el método `getTransmitMTU()` de la clase *L2CAPConnection* para obtener el valor del MTU saliente y así evitar que se envíen demasiado datos.
3. La aplicación no especifica el $ReceiveMTU_A$, pero si especifica el $TransmitMTU_A$, en este caso, la aplicación advierte al dispositivo remoto en la solicitud de la configuración que $ReceiveMTU_A$ tiene el valor DEFAULT_MTU (672 bytes); mientras que el manejo del $TransmitMTU_A$ es similar al del caso 1.
4. La aplicación no especifica ni el $ReceiveMTU_A$ ni el $TransmitMTU_A$, para este caso entonces, el manejo de $ReceiveMTU_A$ es similar al del caso 3, y el manejo de $TransmitMTU_A$ es similar al del caso 2.

3.1.13.2 Clases de Conexión L2CAP

Para la conexión L2CAP la especificación proporciona dos interfaz y una clase: *L2CAPConnection*, *L2CAPConnectionNotifier* y *BluetoothConnectionException*, respectivamente, las cuales se encuentran dentro del paquete *javax.bluetooth*.

Interfaz javax.bluetooth.L2CAPConnection

Esta interfaz representa las conexiones L2CAP. Contiene métodos que permiten obtener los MTUs usadas, enviar y recibir datos.

Interfaz javax.bluetooth.L2CAPConnectionNotifier

El único método en esta interfaz es `acceptAndOpen()` el cual se usa por los servidores L2CAP para escuchar las conexiones entrantes del cliente.

Clase javax.bluetooth.BluetoothConnectionException

Esta excepción se ejecuta cuando una conexión Bluetooth (RFCOMM o L2CAP) no se puede establecer con éxito. El método `getStatus()` de esta clase indicará la razón por la que fracaso la conexión.

3.1.14 Protocolo de Intercambio de Objetos (OBEX)

OBEX es un protocolo desarrollado por la Asociación de Datos Infrarrojos (IrDA®[23]) que permite "lanzar" u "obtener" objetos de clientes y servidores. Para efectuar la transferencia del objeto se establece una sesión OBEX, dicha sesión inicia estableciendo una conexión OBEX mediante una solicitud de *conexión*, mientras que para finalizar la sesión basta con una solicitud *desconexión*. Entre las solicitudes de *Conexión* y *Desconexión*, el cliente puede obtener objetos desde el servidor o enviar objetos al servidor, estos objetos pueden ser archivos, vCards (el cual es un formato de datos para tarjetas comerciales electrónicas), arreglos de bytes y así sucesivamente.

OBEX escala fácilmente de pequeños objetos a grandes objetos y esto se logra enviando un objeto en múltiples paquetes OBEX así cuando un cliente emite una solicitud para enviar u obtener un objeto grande, se inicia una operación OBEX y ésta continúa hasta que el objeto completo se envía al servidor, en el servidor el objeto es recuperado completamente, siempre y cuando no ocurra un error; para completar la operación de envío, el cliente divide el objeto en pequeñas partes y las envía individualmente; el cliente no envía una subsecuencia de partes hasta que la parte anterior sea reconocida. La operación de obtención de un objeto trabaja de manera similar, solo que es el

[23] http://www.irda.org

servidor el que secciona el objeto en partes más pequeñas. Generalmente esta organización en paquetes es transparente a la aplicación.

OBEX, así como HTTP, brinda los métodos que permiten pasar información adicional entre cliente y servidor usando cabeceras, a diferencia de las cabeceras de HTTP que son cadenas, las cabeceras OBEX son valores o secuencias de bytes, en las que se incluyen la longitud, nombre, tipo de descripción e incluso cabeceras HTTP-específicas.

3.1.14.1 Apreciación de la API

La API OBEX le permite a cualquier aplicación completar las operaciones OBEX entre un cliente y un servidor, para esta especificacion se soportan las siguientes operaciones OBEX:

- CONNECT
- PUT
- GET
- SETPATH
- ABORT
- CREATE-EMPTY
- PUT-DELETE
- DISCONNEC

Como se mencionó en la sección anterior, los paquetes OBEX consisten en una colección de cabeceras, para esta especificación las cabeceras OBEX listadas en la tabla 3.1 son accesibles.

NOMBRE DE CABECERA	METODOS PARA MANIPULAR CABECERA
Count	`HeaderSet.getHeader(), HeaderSet.setHeader()`
Name	`HeaderSet.getHeader(), HeaderSet.setHeader()`
Type	`HeaderSet.getHeader(), HeaderSet.setHeader()`
Lenght	`HeaderSet.getHeader(), HeaderSet.setHeader()`
Time	`HeaderSet.getHeader(), HeaderSet.setHeader()`
Description	`HeaderSet.getHeader(), HeaderSet.setHeader()`
Target	`HeaderSet.getHeader(), HeaderSet.setHeader()`
http	`HeaderSet.getHeader(), HeaderSet.setHeader()`
Body	`Operation.openInputStream(), Operation.openDataInputStream(), Operation.openOutputStream(), Operation.openDataOutputStream()`
End of Body	`Operation.openInputStream(), Operation.openDataInputStream(), Operation.openOutputStream(), Operation.openDataOutputStream()`
Who	`HeaderSet.getHeader(), HeaderSet.setHeader()`
Connection ID	`ClientSession.setConnectionID(), ClientSession.getConnectionID(), ServerRequestHandler.setConnectionID(), ServerRequestHandler.getConnectionID()`

Application Parameters	`HeaderSet.getHeader(), HeaderSet.setHeader()`
Authentication Challenge	`HeaderSet.createAuthenticationChallenge(), Authenticator.getPasswordAuthentication()`
Authentication Response	`Authenticator.getPasswordAuthentication(), Authenticator.validatePassword()`
Object Class	`HeaderSet.getHeader(), HeaderSet.setHeader()`
User Define	`HeaderSet.getHeader(), HeaderSet.setHeader()`

Tabla 2.2. Cabeceras OBEX soportadas por esta especificación.

Existen dos modelos de multiplexacion diferentes, los cuales se realizan en la capa de transporte.

Las siguientes excepciones se pueden ejecutar con una llamada al método `Connector.open()`.

- *ConnectionNotFoundException* - se ejecuta cuando el esquema usado en la cadena de conexión no es valido o cuando el tipo de protocolo no existe.
- *IllegalArgumentException* – se ejecuta cuando no se reconocen los parámetros de la cadena de conexión.
- *IOException* – se ejecuta cuando no se puede establecer una conexión.

3.1.14.2 Conexión del cliente

Para crear una conexión cliente en OBEX, la aplicación cliente utiliza una cadena de conexión, que se describirá mas adelante, y la pasa al método `Connector.open();` este método devuelve un objeto *javax.obex.ClientSession*.

Para establecer una conexión OBEX, el cliente crea un objeto *javax.obex.HeaderSet* usando el método `createHeaderSet()` en la interfaz *ClientSession*. Empleando el objeto *HeaderSet*, el cliente puede especificar los valores de la cabecera para una solicitud CONNECT. Un paquete CONNECT OBEX también contiene la versión del OBEX, banderas y la longitud máxima del paquete la cual se conserva al ejecutar la aplicación; para completar la solicitud CONNECT, el cliente proporciona el objeto *HeaderSet* al método `connect()` en la interfaz *ClientSession*; una vez que finaliza la solicitud CONNECT, la cabecera OBEX recibida en el servidor regresa a la aplicación. Si el objeto cabecera no se proporciona como parámetro de entrada, se retorna un objeto *javax.obex.HeaderSet* desde el método `connect()`. Para determinar si la solicitud fue exitosa, el cliente invoca al método `getResponseCode()` en la interfaz *HeaderSet*, el cual devuelve el código de respuesta enviado por el servidor y definido en la clase *javax.obex.ResponseCodes*.

La solicitud de DISCONNECT se completa de la misma manera que la solicitud de CONNECT excepto que en lugar del método `connect()` se llama al método `disconnect()`. Cuando un objeto *javax.obex.HeaderSet* contiene más cabeceras de las que permite un paquete OBEX, se ejecuta una *java.io.IOException*.

Cuando se desea completar una operación SETPATH, el cliente llama al método `setPath()` en el objeto *ClientSession*, para especificar el nombre del directorio target[24], se fija el nombre de la cabecera con el target deseado invocando al método `setHeader()` en el *HeaderSet* proporcionado por el método `setPath()`. El cliente también puede especificar si el servidor debe hacer un back up del directorio antes de especificar el nombre y si el servidor debe crear dicho directorio en caso de que no exista. Si la cabecera es demasiado grande para enviarla en un paquete OBEX, se ejecuta una *java.io.IOException*.

Para completar una operación PUT o GET, el cliente crea un objeto *javax.obex.HeaderSet* con `createHeaderSet()`, luego de especificar el valor de la cabecera el cliente invoca al método `put()` o `get()` en el objeto *javax.obex.ClientSession*, entonces la aplicación envía las cabeceras al servidor y recibe como respuesta el objeto *javax.obex.Operation* de los métodos `put()` y `get()`; con éste objeto, el cliente puede determinar si la solicitud tuvo éxito. Cuando la solicitud es exitosa, el cliente puede lanzar u obtener un objeto de datos usando los flujos de salida o entrada, respectivamente; dichos flujos se terminan cuando el cliente finaliza la conexión. Para solicitudes ABORT PUT o ABORT GET, el cliente invoca el método `abort()` en el objeto *javax.obex.Operation*, este método cierra todos los flujos de entrada y salida y además finaliza todas las operaciones llamando al método `close()` en el objeto *Operation*.

3.1.14.3 Conexión del servidor

Para crear una conexión, el servidor le proporciona una cadena de conexión al método `Connector.open()`, este método devuelve un objeto *javax.obex.SessionNotifier*, el cual espera por un cliente para crear una conexión en la capa de transporte llamando al método `acceptAndOpen()`, cuando dicho método se invoca muchas veces permite que el servidor establezca múltiples conexiones con múltiples clientes. El método `acceptAndOpen()` devuelve un objeto *javax.obex.Connection*, este objeto representa una conexión con un solo cliente. El servidor especifica al gestor de la solicitud que responderá a dicha solicitud OBEX del cliente pasándole el objeto *javax.obex.ServerRequestHandler* al método `acceptAndOpen()`.

[24] Directorio Target se refiere a un directorio designado o especifico.

En el caso que la aplicación servidor llame al método `abort()` el argumento *javax.obex.Operation*, el cual hace parte de los métodos `onGet()` y `onPut()`, se ejecuta una *java.io.IOException.*

Si la aplicación servidor no puede pasar todas las cabeceras que se especifican para una respuesta, entonces la aplicación servidor devuelve *OBEX_HTTP_REQ_TOO_LARGE*. Si la aplicación servidor retorna un código de respuesta que no esta definido en la clase *javax.obex.ResponseCodes*, entonces, la aplicación servidor le envía una respuesta *OBEX_HTTP_INTERNAL_ERROR* al cliente.

3.1.14.4 Descripción de la Cadena de conexión

Para crear un objeto de conexión cliente o servidor en OBEX, la aplicación usa el siguiente formato:
{*protocol*}:[{*target*}] [{*params*}]

La definición de {*protocol*}, {*target*}, y {*params*} depende de la capa de transporte que OBEX emplee, pero en general, {*protocol*} se define para que sea {*transport*} obex.

Estos protocolos deben ser implementados basados en el mecanismo de transporte disponible en los dispositivos. Por ejemplo, para un dispositivo que soporta un puerto infrarrojos, entonces sólo es necesario que se implemente el protocolo *"irdaobex"*. Cuando se llama el método `Connector.open()` en un protocolo de transporte que no ofrezca soporte, entonces se ejecuta una *ConnectionNotFoundException*.

3.1.14.5 OBEX sobre RFCOMM

El parámetro {*protocol*} para OBEX sobre RFCOMM se define como *btgoep* debido a que ésta es la implementación del Perfil de Intercambio de Objetos Genérico (GOEP). El {*target*} para la conexión con el cliente es la dirección Bluetooth y el identificador del canal del dispositivo que el cliente desea conectar, estos dos valores deben ir separados por dos puntos (por ejemplo, *0050C000321B:4*). Para un servidor el {*target* } siempre es *localhost* seguido por dos puntos y el UUID de la clase de servicio. Los {*params*} válidos para OBEX sobre RFCOMM son *authenticate*, *encrypt*, *authorize*, y *master*. El valor por defecto para todos éstos {*params*} es false, sin embargo true es el único valor válido.

La siguiente es una cadena de conexión cliente válida para OBEX sobre RFCOMM:
`btgoep://0050C000321B:12`

La siguiente es una cadena de conexión servidor válida para OBEX sobre RFCOMM:
`btgoep://localhost:12AF51A9030C4B2937407F8C9ECB238A`

Cuando una aplicación pasa una cadena de conexión servidor valida al método `Connector.open()`, se crea un registro de servicio Bluetooth.
Los objetos representan un servicio OBEX cuando:

1. Un objeto implementa la interfaz *javax.obex.SessionNotifier* y escucha las conexiones del cliente a este servicio.
2. Un objeto implementa la interfaz *ServiceRecord* y describe tanto el servicio como sus parámetros de conexión al dispositivo del cliente.

3.1.14.6 OBEX sobre TCP/IP

Cuando OBEX utiliza a TCP/IP como protocolo de transporte, el {*protocol*} es *tcpobex*. Para un cliente OBEX el {*target*} es la dirección IP del servidor seguido por dos puntos y el número del puerto por ejemplo (12.34.56.100:5005), cuando no se especifica el número del puerto, se usa el puerto número 650 (éste es el número del puerto reservado para OBEX por IANA, la Autoridad de Números Asignados para Internet). Para el servidor el {*target*} tiene dos puntos seguido por el número del puerto por ejemplo (:5005), en el caso de que no se especifique el puerto, por defecto se asigna el puerto número 650. Cuando se utiliza OBEX sobre TCP/IP no se definen {*params*} validos.

Las siguientes son cadenas de conexión cliente válida para OBEX sobre TCP/IP:
`tcpobex://132.53.12.154:5005`
`tcpobex://132.53.12.154`
La primera cadena crea a un cliente que se conecta al puerto 5005, mientras que la segunda cadena crea a un cliente que se conecta al puerto 650.

Las siguientes son cadenas de conexión servidor validas para OBEX sobre TCP/IP:
`tcpobex://:5005`
`tcpobex://`
La primera cadena crea un servidor que escucha en el puerto 5005, mientras que la segunda cadena crea un servidor que escucha en el puerto 650.

3.1.14.7 OBEX sobre IrDA

En el caso de que OBEX utilice a IrDA como protocolo de transporte, el {*protocolo*} es *irdaobex*. Para los clientes OBEX, el {*target*} inicia con los parámetros *discover*, *addr*, *conn*, o *name* seguidos

por los parámetros adicionales cuando sea necesario, mientras que para los servidores OBEX, el {*target*} inicia con *localhost*.

3.1.14.7.1 Identificador de Descubrimiento de dispositivo

Cuando {*target*} tiene como primer parámetro *discover*, la pila de protocolos IrDA inicia un descubrimiento de dispositivos para determinar cuales dispositivos infrarrojos están dentro del rango, si se descubre más de un dispositivo, la aplicación intenta conectarse a cada uno de ellos de manera exitosa y la solicitud de servicio se completa; cuando se descubren dispositivos no permitidos, el proceso de descubrimiento se repite por un periodo de tiempo especificado por la aplicación antes de reportar un fracaso. Si existe previamente una lista de dispositivos descubiertos, la implementación puede intentar conectarse a esos dispositivos, sin embargo, si el intento de conexión fracasa, antes de reportar alguna falla la implementación deben intentar de nuevo hacer un descubrimiento.

discover puede estar seguido por un "**.**" y una representación hexadecimal multi-byte del servicio requerido, esta representación se obtiene durante el proceso de descubrimiento. Los bits proporcionados de esta forma, se denominan comúnmente bits de indicación y se usan para limitar los intentos de conexión a tan sólo los dispositivos para los cuales indiquen estos bits. Se debe tener cuidado cuando se especifican los bits de indicación en la conexión del cliente. Los bits de indicación no son un medio fiable para determinar el tipo de dispositivo o sus servicios disponibles, empleando ciertos bits de indicación, las aplicaciones podrían limitar la interoperabilidad con dispositivos remotos que, por alguna razón, tienen fallas para fijar los bits de indicación.

3.1.14.7.2 Identificador Target para Servidores OBEX

Para indicar la disponibilidad de un servicio, el parámetro {*target*} inicia con *localhost* seguido por un conjunto de bits de indicación que usan el mismo mecanismo descrito en la sección anterior para el descubrimiento. Primero se adicionan los bits de indicación en el dispositivo servidor, varias aplicaciones pueden especificar el mismo bit de indicación, el cual permanecerá fijo hasta que el último servicio se cierre. Para OBEX el bit de indicación por defecto es 0200 y se establece automáticamente al abrir una conexión servidor *irdaobex.* Además, no resulta necesario que se especifique explícitamente en la aplicación.

3.1.14.7.3 Identificación del servicio

OBEX sobre IrDA permite definir los nombres de las clases IAS[25] en la cadena `Connector.open()` a través de la sección {*params*}. Los {*params*} tienen el nombre "ias" y además tienen como valor la lista con los nombres de las clases IAS, generalmente los nombres de las clases individuales están separados por "**,**".

Por ejemplo, la siguiente cadena de conexión:
`irdaobex://discover;ias=MyAppOBEX,OBEX,OBEX:IrXfer;`
especifica que la aplicación además de descubrir los dispositivos debe intentar consultar los servicios basándose en los nombres de las clases IAS de los servicios *MyAppOBEX*, *OBEX*, y *OBEX:IrXfer*.

Si la lista con los nombres de servicio no se especifica, se utilizan los dos nombres de servicios predefinidos en OBEX, los cuales son *OBEX* y *OBEX:IrXfer*.

3.1.14.8 Autenticación

Para autenticar un cliente o servidor en OBEX, tanto el cliente como el servidor deben compartir una contraseña secreta, esta contraseña en realidad nunca se envía o intercambia como parte del procedimiento de autenticación OBEX; cuando el cliente desea autenticar al servidor, envía una cabecera con un desafío de autenticación al servidor, dicho desafío contiene 16 bytes, cuando el servidor recibe esta cabecera determina la contraseña correcta, luego combina la contraseña con el desafío y aplica un algoritmo. El algoritmo resultante, se denomina resumen de respuesta y se envía en la cabecera de autenticación de respuesta. Cuando el cliente recibe esta cabecera debe determinar que contraseña es, para ello combina el desafío y entonces envía la cabecera con la contraseña correcta Finalmente el resumen resultante se compara con el resumen recibido en la cabecera de respuesta y si el resultado es el mismo, el servidor se ha autenticado. Para autenticar al cliente el servidor realiza un proceso similar.

El proceso de autenticación inicia con una llamada al método `createAuthenticationChallenge()`, este método le dice a la implementación que incluya una cabecera con un desafío de autenticación en la próxima solicitud o contestación, y también le permite a la aplicación proporcionar una descripción de la contraseña que se debe usar, el tipo de acceso y si es necesario un nombre de usuario; la aplicación esta encargada de generar el desafío.

[25] En este nivel es donde se buscan y encuentran los diferentes dispositivos infrarrojos

Para facilitar el proceso de autenticación, la interfaz *Authenticator* proporciona métodos que se pueden implementar por una aplicación para responder a un desafío de autenticación y a cabeceras de autenticación. Cuando una cabecera con un desafío de autenticación se recibe, se hace una invocación del método `onAuthenticationChallenge()`, el cual proporciona la descripción junto con alguna información adicional. Para este caso el desafío no lo brinda la aplicación pero en cambio, la aplicación espera proporcionar el nombre de usuario correcto y contraseña a través de un objeto *PasswordAuthentication* devolviendo este objeto del método `onAuthenticationChallenge()`.

Cuando una respuesta de autenticación se recibe, si el nombre de usuario viene en la cabecera de respuesta de autenticación, se hace un llamado al método `onAuthenticationResponse()` utilizando el nombre de usuario y para este caso la aplicación debe entonces determinar cual es la contraseña correcta y devolver la contraseña al método `onAuthenticationResponse()`; luego se combina la contraseña devuelta con el desafío enviado en la cabecera de autenticación y se le aplica un algoritmo. La aplicación compara el resumen de la respuesta recibida en la cabecera de autenticación de respuesta y el algoritmo producido, si los valores no son iguales y la solicitud de autenticación generada por un cliente OBEX se realiza mediante una llamada a `connect()`, `setPath()`, `delete()`, `get()`, `put()`, o `disconnect()`, entonces se ejecuta una *IOException*. Alternativamente un cliente OBEX puede generar la solicitud de autenticación llamando al método `createAuthenticationChallenge()` en un objeto *HeaderSet* que se pasa a un objeto *Operation* a través de su método `sendHeaders()`, cuando los valores no son iguales, se ejecuta una *IOException* después de cualquier llamada subsecuente a cada objeto *Operation* o cualquier flujo construido por el mismo objeto *Operation*. En el caso del servidor OBEX, si los valores no son iguales, se llama al método `onAuthenticationFailure()` en el servidor *ServerRequestHandler*. Una *IOException* se ejecutará después de cualquier llamada subsecuente por el servidor a cada objeto *Operation* asociado con esta conexión OBEX o cualquier flujo construido por el mismo objeto *Operation*.

3.1.14.9 Clases OBEX

Para OBEX la especificación proporciona cinco interfaces y tres clases: *ClientSession, HeaderSet, ResponseCodes, ServerRequestHandler, SessionNotifier, Operation, Autenticador y PasswordAuthentication* las cuales se encuentran dentro del paquete *javax.obex*.

Interfaz javax.obex.ClientSession

Representa un objeto de conexión en el lado del cliente para OBEX. Proporciona los métodos para las siguientes operaciones *CONNECT*, *DISCONNECT*, *SETPATH*, *PUT-DELETE*, *CREATE-EMPTY*, *PUT* y *GET*.

Interfaz javax.obex.HeaderSet

Define las cabeceras OBEX que se pueden establecer en una operación. Proporciona los métodos *get* y *set* para todas las cabeceras OBEX además, los clientes pueden crear un objeto *HeaderSet* llamando al método `createHeaderSet()` dentro del objeto *javax.obex.ClientSession*.

Clase javax.obex.ResponseCodes

Define el código de respuesta valido para un servidor OBEX.

Clase javax.obex.ServerRequestHandler

Define la manera de cómo manejar solicitudes de un cliente OBEX. La aplicación que extiende de esta clase necesita sustituir sólo esos métodos para las solicitudes que soporta el cliente.

Interfaz javax.obex.SessionNotifier

Define el objeto notificador de sesión del servidor el cual se devuelve de una llamada al método `Connector.open()` para conexiones con el servidor. Además, proporciona los métodos para esperar que un cliente establezca la conexión en la capa de transporte.

Interfaz javax.obex.Operation extiende

Define un objeto *operación* que se usa para operaciones *PUT* y *GET*. Las operaciones OBEX se ejecutan automáticamente y sin problemas para la aplicación, porque los paquetes son leídos y escritos por la implementación de la API. Esta interfaz también proporciona un método para la operación *ABORT*.

Interfaz Autenticador

Se ocupa de los problemas relacionados con la autenticación y cabeceras de respuesta de autenticación.

Clase PasswordAuthentication

Encapsula un nombre de usuario y la contraseña para autenticación.

3.2 API C ++ BASADO EN EL S.O. SYMBIAN PARA BLUETOOTH

3.2.1 Introducción

Esta especificación que tiene como plataforma base el SO Symbian[26] y que usa como lenguaje de programación C++. Tiene como objetivo principal dar soporte a la tecnología de comunicaciones de corto alcance Bluetooth, brindado una guía completa de los procedimientos paso a paso para el uso de las APIs que ofrece.

3.2.2 Descripción de la Especificación C++

La funcionalidad de esta especificación es proporcionada por varias APIs que son usados de la siguiente forma:

- *Sockets Bluetooth*
  Los Sockets Bluetooth encapsulan el acceso a capas del stack de protocolos Bluetooth L2CAP y RFCOMM a través de una interfaz de sockets como TCP/IP.

- *Base de Datos del Descubrimiento del Servicio Bluetooth*
  La Base de Datos del Descubrimiento del Servicio encapsula parte del SDP: un servicio local usa este API para registrar sus atributos, y por tanto para que dispositivos remotos puedan descubrir su presencia y determinar si es apropiado.

- *Agente del Descubrimiento del Servicio Bluetooth*
  El Agente del Descubrimiento del Servicio encapsula la otra parte del SDP: este permite descubrir los servicios que están disponibles en un dispositivo remoto, y los atributos de dichos servicios.

- Gestor *de Seguridad Bluetooth*
  El Gestor de Seguridad habilita a los servicios para que puedan configurar los requerimientos de seguridad que las conexiones entrantes deben reunir.

3.2.3 Arquitectura de la Especificación C++

La Figura 3.6 muestra las relaciones que existen entre las APIs que ofrece esta especificación. Como se puede observar, la API fundamental en la cual las otras APIs se soportan para el

[26] Symbian es un sistema operativo para dispositivos móviles. Mayor información http://www.symbian.com/about/symb-os.html

establecimiento de comunicaciones con otros dispositivos es la API Sockets Bluetooth.

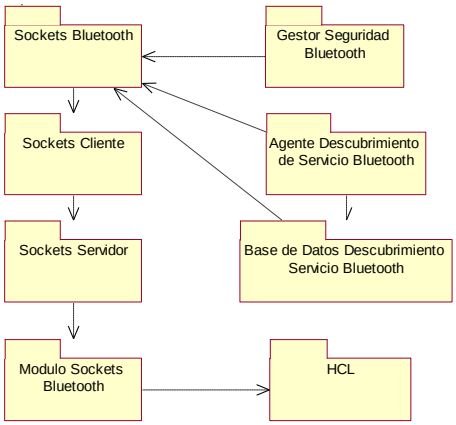


Figura 3.6 Arquitectura APIs Bluetooth – Especificación C++

3.2.3.1 Sockets Bluetooth

Este API tiene como propósito brindar las capacidades para que dispositivos Bluetooth puedan descubrir otros dispositivos Bluetooth remotos, y además para que sobre el enlace establecido se pueda leer y escribir datos. Hay dos roles importantes en el proceso de establecimiento de un canal de comunicaciones sobre Bluetooth: iniciador y receptor. El receptor debe esperar por un iniciador para establecer la conexión. Una vez la conexión se ha establecido, cualquiera de los dos terminales puede enviar y recibir datos o terminar la conexión. Este API permite que cualquiera de los dos roles sea programado.

3.2.3.1.1 Descripción

El API Sockets Bluetooth define los siguientes conceptos: dirección del socket e indagación al dispositivo remoto.

- *Dirección del Socket*

Esta dirección corresponde a la dirección única de 48-bits que tiene cada dispositivo Bluetooth.

- *Indagación al dispositivo remoto*

Un cliente puede indagar por dispositivos remotos disponibles a través de la clase sockets `RHostResolver`, esta clase ofrece una interfaz que tiene las siguientes capacidades: obtener nombres de los dispositivos remotos a partir de sus direcciones, obtener direcciones a partir de los nombre y obtener o establecer nombres al dispositivo local.

3.2.3.1.2 Cómo encontrar dispositivos remotos y establecer una conexión

Existen varios procedimientos que una aplicación tiene que llevar a cabo cuando desea establecer una conexión con un dispositivo remoto y posteriormente utilizar un servicio que dicho dispositivo ofrece. En primera instancia se debe determinar el dispositivo con el cual el usuario desea establecer una comunicación, a continuación la aplicación averigua si el servicio que se solicita esta disponible y finalmente establece la comunicación, es posible que se ejecute algún tipo de seguridad si es necesario. Este proceso se explicará más detalladamente posteriormente.

- *Métodos para seleccionar un dispositivo remoto*

Una aplicación puede determinar a que dispositivo remoto desea conectarse de diferentes formas: recuperando la dirección del dispositivo remoto desde la base de datos local en la cual se ha almacenado algunas direcciones de preferencia; a partir de una selección que haga el usuario utilizando la UI Bluetooth o partir de algún criterio en particular programado en la aplicación por ejemplo “seleccionar el punto de acceso a LAN menos congestionado”.

- *Cómo hacer indagaciones a dispositivos remotos*

Cada dispositivo Bluetooth tiene una dirección única de 48 bits dentro de su hardware, es por esto que una indagación básica dentro de un rango de cobertura retornaría este tipo de direcciones. Sin embargo, un dispositivo Bluetooth también puede tener un nombre “amistoso” en caso que se desee desplegar una lista de dispositivos disponibles al usuario. Además si el hardware lo soporta se podría hacer indagaciones de direcciones y nombres simultáneamente.

Las indagaciones de direcciones y nombres se llevan a cabo a través de la clase sockets del SO Symbian `RHostResolver`. También se utiliza la clase `TInquirySockAddr`, porque esta encapsula las direcciones Bluetooth y consulta las clases de dispositivo y servicio.

Para obtener las direcciones de los dispositivos remotos se deben seguir los siguientes pasos:

1. Se crea un objeto Socket Servidor a partir de la clase `RSocketServ y se conecta a` este, también se selecciona el protocolo que se va a utilizar, así:

```
RSocketServ socketServ;
socketServ.Connect()
socketServ.FindProtocol(KL2Cap);
```

2. Crear e inicializar un objeto `RHostResolver` a través del cual se realizaran las consultas.

```
RHostResolver hr;
hr.Open()
```

3. Para especificar si se va a realizar una búsqueda de direcciones o nombres de dispositivos Bluetooth, se crea un objeto `TInquirySockAddr` que a partir de la función `SetAction()` puede ser fijado en una constante `KHostResInquiry` o `KHostResName` para búsqueda de direcciones o nombres respectivamente, teniendo esto la consulta puede iniciarse haciendo uso del método `GetByAddress()`, a continuación se muestre este proceso:

```
TInquirySockAddr addr;
addr.SetAction(KHostResInquiry);
hr.GetByAddress(addr);
```

4. Para obtener todos los dispositivos descubiertos, se hace una llamada repetida al método `Next()` hasta que se retorne un error; `hr.Next();`

Para obtener los nombres de los dispositivos remotos se siguen los mismos pasos anteriores pero teniendo en cuenta que el objeto `TInquirySockAddr` se fija en la constante `KHostResName`.

- *Cómo hacer indagaciones de servicios en un dispositivo remoto*
- Un dispositivo remoto puede tener más de un proveedor de servicios Bluetooth y estos servicios ofrecidos en la mayoría de los casos pueden ser leídos desde la clase del dispositivo. Esta especificación define que La clase del dispositivo se obtiene a partir de la función `MajorClassOfDevice()` de la clase `TInquirySockAddr` una vez realizada la indagación de la dirección del dispositivo. Este proceso se explica con mas detalle en el API *Agente del Descubrimiento del Servicio Bluetooth*

- *Cómo conectarse y transferir datos a un dispositivo remoto*

Una vez el dispositivo y el servicio se han establecido, se puede realizar la conexión con el servicio remoto y hacer uso de éste, a través de la función `Connect()` de la clase socket `RSocket` que además define funciones para la creación de sockets, escritura y lectura de datos sobre los mismos. Los datos pueden estar en cualquier formato siempre y cuando el servicio designado los entienda. (comandos AT, texto, HTTP, PPP, etc).

3.2.3.1.3 Cómo escuchar las solicitudes de conexión desde dispositivos remotos

Para que un dispositivo esté en capacidad de recibir conexiones entrantes, dentro de la aplicación del servicio de debe abrir un socket que permanentemente escuche (socket listening), sobre el cual se puedan aceptar las conexiones; los pasos necesarios son los siguientes:

1. Crear un objeto de la Clase `RSocketServ`, conectarse a dicho Socket Servidor y seleccionar el protocolo que va a ser usado. Mirar el código de ejemplo

```
RSocketServ socketServ;
socketServ.Connect();
 _LIT(KL2Cap, "L2CAP"); //o RFCOMM dependiendo del que se va a usar
```

2. Abrir un socket para ese protocolo

```
RSocket listen;
listen.Open();
```

3. Disponer al socket para que escuche las conexiones entrantes usando la función Listen() de la clase RSocket.

```
listen.Listen();
```

4. Cuando el receptor desea cerrar la conexión, debe asegurarse de cerrar el socket que esta escuchando además de los sockets conectados.

Es importante tener en cuenta que antes de aceptar conexiones entrantes es necesario fijar los requerimientos de seguridad del servicio y advertir la disponibilidad del mismo.

3.2.3.1.4 Cómo obtener o establecer el nombre de un dispositivo local

Cada dispositivo Bluetooth tiene un nombre que les permite a los usuarios identificarlo. El nombre del dispositivo local se obtiene y se establece a través de las funciones `GetHostName()` y `SetHostName()` respectivamente. Estas funciones son definidas en la clase `RHostResolver`.

```
// De esta manera se crea una instancia de la clase RHostResolver
   RHostResolver hr;
```

```
// Para obtener el nombre del dispositivo local
   TInt ret =hr.GetHostName();
```

3.2.3.2 Base de Datos del Descubrimiento del Servicio Bluetooth

Esta API, tiene como finalidad permitir a un servicio local ingresar sus propiedades a una base de datos del servicio Bluetooth local y con esto habilitar a los servicios Bluetooth remotos para que descubran que dicho servicio esta disponible, esta API permite el uso del Protocolo de Descubrimiento del Servicio Bluetooth.

3.2.3.2.1 Registros y atributos del servicio

Toda la información acerca de un servicio esta contenida dentro de un solo registro de servicio, por lo tanto un registro de servicio consiste prácticamente en una lista de atributos del servicio.

Cada atributo esta conformado por un ID, un tipo y un valor; los tipos y algunos IDs de atributos están predefinidos.

Uno de los atributos más importantes es la *Clase de Servicio,* al ser este una instancia de una clase de servicio proporciona las capacidades del servicio; como una misma aplicación puede proporcionar múltiples servicios, cada uno de ellos debe tener un registro de servicio diferente y estos a su vez una lista de atributos de clases de servicios. Por ejemplo un mismo servicio puede indicar a través de una clase de servicio que proporciona el servicio de impresión y por medio de otra clase de servicio que proporciona el servicio de impresión Postscript.

Las clases de servicio en un registro de servicio son especificadas usando unos Identificadores Únicos Universales (Universally Unique Identifier – UUID). Para mayor información sobre los valores predefinidos de los IDs de las clases y los atributos ver *Documento Bluetooth Assigned Numbers*[27].

3.2.3.2.2 Cómo conectarse a la base de datos de descubrimiento del servicio

Para que un cliente pueda hacer uso de la base de datos del descubrimiento del servicio debe realizar lo siguiente:

1. Crear un objeto de la clase RSdp, esta clase proporciona una sesión a la base de datos del descubrimiento del servicio, y conectarse a ésta a través del método `Connect()` que proporciona esta clase.

   ```
   RSdp sdp;
   sdp.Connect();
   ```

[27] Se encuentra en la dirección http://www.bluetooth.org/assigned-numbers/

2. Crear y conectarse a una subsesión del objeto de la base de datos RSdp, a través de la cual los servicios y atributos podrán ser adicionados, actualizados o eliminados. La subsesión se crea a través de la clase `RSdpDatabase`.

```
RSdpDatabase sdpSubSession;
sdpSubSession.Open(sdp));
```

3. Cerrar las sesiones y subsesiones cuando no son requeridas.

```
sdpSubSession.Close();
sdp.Close();
```

3.2.3.2.3 Cómo crear y manipular atributos

Los atributos en los registros de servicio pueden ser de uno o varios tipos. Este API ofrece una familia de clases que encapsula y permite la manipulación de cada tipo de atributo, y son derivadas de la clase base `CSdpAttrValue`. El tipo es fácilmente deducible a partir del nombre de la clase: por ejemplo `CSdpAttrValueUint` encapsula un atributo de tipo entero sin signo. Algunas de las clases que se derivan son las siguientes:

- CSdpAttrValueBoolean_encapsula atributos de tipo Boolean
- CSdpAttrValueInt_ encapsula atributos de tipo Entero
- CSdpAttrValueString_ encapsula atributos de tipo String
- CSdpAttrValueUint_encapsula atributos de tipo Entero sin Signo
- CSdpAttrValueURL_encapsula atributos de tipo URL

3.2.3.2.4 Cómo registrar un servicio en la base de datos

Una vez creada la sesión y la subsesión con la base de datos, se puede crear un registro de servicio siguiendo estos pasos:

1. Crear un registro de servicio vacío a partir de un objeto de la clase `TSdpServRecordHandle` de esta forma:

```
TSdpServRecordHandle recordHandle = 0;
```

2. Como el atributo clase de servicio será un UUID, se debe crear y definir un objeto `TUUID` con el valor predefinido, esto para crear la clase de servicio, así:

```
TUUID uuid(0x20000);
```

3. Adicionar el registro del servicio a la base de datos. A partir de la subsesión que ya ha sido creada y abierta, se crea el registro del servicio con la función

`CreateServiceRecordL()` suministrando los parámetros del registro y la clase del servicio, así:

```
sdpSubSession.CreateServiceRecordL(uuid, recordHandle);
```

3.2.3.2.5 Cómo fijar un atributo en un registro de servicio

Un atributo de servicio puede ser fijado o actualizado llamando a la función `UpdateAttributeL()` a partir de la subsesión a la base de datos que ha sido creada y abierta. En el siguiente ejemplo se va a adicionar un atributo con ID 102 hex en el registro de servicio identificado por `recordHandle`:

```
sdpSubSession.UpdateAttributeL(recordHandle, 0x0102);
```

3.2.3.3 Agente del Descubrimiento del Servicio Bluetooth

Este API tiene objetivo brindar las capacidades para realizar el descubrimiento de los servicios y atributos que se encuentran disponibles en un dispositivo remoto.

3.2.3.3.1 Cómo Indagar por un servicio remoto

El Agente de Descubrimiento del Servicio como se mencionó se utiliza para hacer indagaciones de servicios Bluetooth que se encuentran disponibles en un dispositivo remoto específico, este procedimiento se puede hacer una vez que los dispositivos que están dentro del rango de cobertura han sido identificados a través del API Sockets Bluetooth. Para que el usuario pueda recibir las respuestas de las indagaciones se debe implementar la interfaz `MSdpAgentNotifier`.

Los pasos para realizar una búsqueda de servicio son los siguientes:

1. Se debe crear un objeto de la clase `CSdpAgent`, a través del cual el Protocolo de Descubrimiento de Servicio (SDP) hace solicitudes a un dispositivo remoto, pasando como parámetro el objeto que implemente la interfaz `MSdpAgentNotifier` y la dirección del dispositivo remoto Bluetooth, de esta manera:

   ```
   CSdpAgent* agent = CSdpAgent::NewLC(rcvr, devAddr); //se asume que rcvr implementa MSdpAgentNotifier
   ```

   y que `devAddr` es la dirección del dispositivo remoto
2. Crear un objeto `CSdpSearchPattern que es una lista p`ara especificar las clases de servicio que se van a buscar. Las clases se pueden adicionar a través de la función `AddL()`, así:

```
CSdpSearchPattern* list = CSdpSearchPattern::NewL();
list->AddL(0x0100);
```

3. Establecer el patrón de búsqueda sobre el objeto `CSdpAgent` usando la función `SetRecordFilterL()`.
   ```
   agent->SetRecordFilterL(*list);
   ```

4. A partir del objeto `CSdpAgent` llamar a la función `NextRecordRequestL()` para obtener los resultados de la búsqueda.
   ```
   agent->NextRecordRequestL();
   ```

3.2.3.3.2 Cómo leer atributos del servicio remoto

Las consultas de atributos de servicios se pueden limitar especificando atributos o un rango de IDs, al igual que en la búsqueda de servicios se debe implementar la interfaz para recibir las respuestas de dichas búsquedas.

Los pasos que se deben seguir para leer los atributos de servicios son los siguientes:

1. Crear un objeto `CSdpAttrIdMatchList` que es una lista en el cual especifican los atributos que se desean recuperar, así:
   ```
   CSdpAttrIdMatchList* matchList = CSdpAttrIdMatchList::NewL();
   ```

2. Adicionar los IDs de los atributos a la lista usando la función `AddL()`. Los IDs se involucran con una estructura de tipo `TAttrRange` y de esta manera se puede especificar un atributo simple o un rango de IDs, así:
   ```
   matchList->AddL(TAttrRange(0x102));
   ```

3. Iniciar la consulta usando la función `AttributeRequestL()` a partir del objeto `CSdpAgent` que se asume ya ha sido creado, pasándole como parámetro la lista de atributos específicos, así:
   ```
   agent->AttributeRequestL(serviceHandle, *matchList);
   ```

3.2.3.4 Gestor de Seguridad Bluetooth

El propósito de esta API es permitir que los servicios Bluetooth establezcan los requerimientos de seguridad apropiados que las conexiones entrantes deben reunir. Estos requerimientos pueden ser autenticación, autorización y/o encriptación.

3.2.3.4.1 Cómo conectarse al Gestor de Seguridad

Para usar el gestor de seguridad, un cliente debe:

1. Crear una sesión al gestor de seguridad a través de la clase `RBTMan`, y abrir una conexión, así:

```
RBTMan secMan;
secMan.Connect();
```

2. Crear una subsesion al gestor de seguridad a través de la clase `RBTSecuritySettings`, y abrirla. Un Cliente puede tener múltiples subsesiones abiertas si así lo requiere. (Las sesiones y subsesiones son parte de la arquitectura del SO Symbian para la comunicación entre procesos), de esta forma:

```
RBTSecuritySettings secmanSubSession;
secmanSubSession.Open(secMan));
```

3. Cerrar las subsesiones y sesiones cuando estas ya no se requieran, de esta manera:

```
secmanSubSession.Close();
secMan.Close();
```

3.2.3.4.2 Cómo establecer los requerimientos de seguridad del servicio

Los requerimientos de seguridad que pueden ser establecidos para un servicio son: autenticación, autorización y/o encriptación.

Para fijar los requerimientos de seguridad, siga los siguientes pasos:

1. Se debe crear un objeto de configuración de seguridad a través de la clase `TBTServiceSecurity.`

```
TBTServiceSecurity secSettings();
```

2. A través del objeto de configuración, establecer el tipo de seguridad que se requiere: autenticación, autorización, y/o encriptación, en el siguiente ejemplo se establecen los tres tipos:

```
secSettings.SetAuthentication(ETrue);
secSettings.SetAuthorisation(ETrue);
secSettings.SetEncryption(ETrue);
```

3. Y finalmente a través de la susbsesión establecida con el gestor de seguridad se pasan los requerimientos de seguridad fijados, se utiliza la función `RegisterService, así:`

```
secmanSubSession.RegisterService(secSettings);
```

3.3 API BLUETOOTH DE DIGIANSWER

3.3.1 Introducción

La API para la tecnología inalámbrica Bluetooth desarrollada por la Compañía DIGIANSWER[28] tiene como principal objetivo brindar las herramientas necesarias para el desarrollo software Bluetooth, proporcionando un nivel de abstracción que le permita al programador no involucrarse directamente con los detalles técnicos relacionados con el stack de protocolos Bluetooth y el hardware en el cual corren las aplicaciones.

3.3.2 Arquitectura del API

Bluetooth es una tecnología en constante desarrollo es por esta razón que la API que se desarrolló es un Interfaz de Programación dinámica y adaptable que se construye sobre una arquitectura "pluggable" que permite agregar nuevos módulos dinámicamente al API sin necesidad de crear una nueva versión del mismo, dichos módulos son llamados "plugins".

Es importante tener en cuenta que este API puede ser accedido desde cualquier lenguaje de programación que soporte el *Modelo de Objeto Común Microsoft* (Microsoft Common Object Model – Microsoft COM)[29], incluyendo lenguajes de programación ampliamente usados como C++, Delphi, y Visual Basic. Con el uso de COM se quiere que la mayoría de los problemas complejos que se ven involucrados con la realización de comunicaciones entre procesos, la comunicación de parámetros de funciones y resultados encuentren solución.

3.3.2.1 Módulos en la API

La Figura 3.7 muestra los diferentes módulos que tiene la API en este momento, el propósito y funcionalidad de los mismos es descrita a continuación:

[28] http://www.digianswer.com/

[29] Microsoft COM es un framework para crear y usar componentes. Un componente puede ser una clase simple que realiza una tarea específica, esta clase puede ser incluida en COM y reutilizada desde cualquier otro lenguaje de programación que soporte COM.

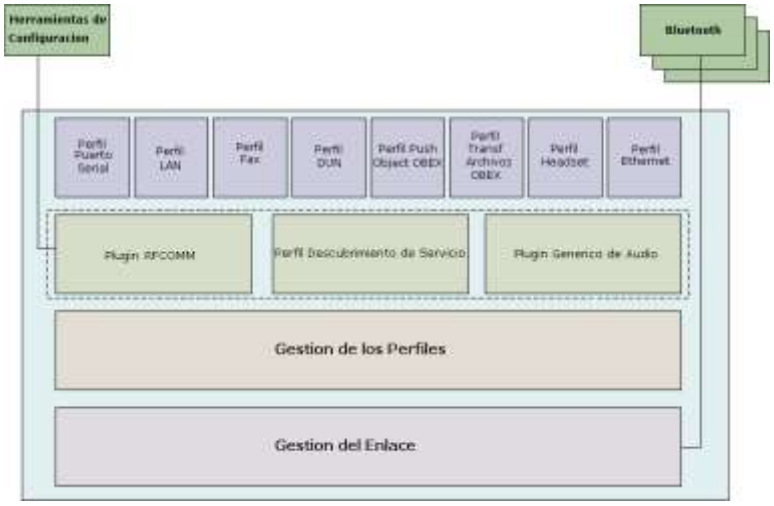


Figura 3.7 Módulos del API - DIGIANSWER

- *Gestor del Enlace*

El módulo Gestor del Enlace es el principal en la API; ya que incluye las capacidades que permiten el establecimiento del enlace Bluetooth, el descubrimiento de dispositivos, y varias configuraciones a nivel de enlace tales como: la seguridad, encripción y el nombre del dispositivo local.

- *Gestor del perfil*

El módulo Gestor del Perfil maneja las diferentes implementaciones de perfiles incluidas en el API y tiene como tarea principal cargar e inicializar dichas implementaciones las cuales se incluyen exclusivamente como “plugins”. Este módulo se invoca durante la inicialización del API y se asegura que todos los módulos del perfil estén corriendo antes de proceder.

- *Descubrimiento del servicio*

El modulo Protocolo de Descubrimiento del Servicio (SDP) se utiliza para almacenar toda la información concerniente a los servicios que están disponibles en un dispositivo y además para obtener información de otros dispositivos. Los módulos de niveles superiores, así como los perfiles, interactúan con este módulo para proporcionar información de servicios a sus clientes.

- *RFCOMM*

Como se mencionó antes, RFCOMM es un término de Bluetooth que representa la transferencia de datos seriales sobre un enlace Bluetooth ofreciendo una emulación del puerto serial. Los canales de datos del puerto serial que están definidos en RFCOMM son manejados por un driver del puerto

COM virtual, que le permitirá a las aplicaciones acceder a esos canales directamente como puertos COM de Windows estándar.

Los siguientes módulos que corresponden a perfiles Bluetooth están implementados como plugins:

- *Audio Genérico*

El plugin de Audio Genérico se ocupa de las conexiones de audio con los dispositivos Bluetooth remotos, estas conexiones normalmente son manejadas por el módulo Gestor del Enlace, sin embargo la razón para mantener la funcionalidad de audio en un plugin separado, es que este plugin además de proporcionar la funcionalidad para establecimiento de una conexión de audio maneja dos modos de operación:

PC Speaker/Speaker Phone: Este modo enruta el sonido desde una conexión de audio Bluetooth a un dispositivo de audio existente, típicamente una tarjeta de sonido. De esta manera, pueden usarse el micrófono y los parlantes de un computador como dispositivos de entrada y salida de audio para una conexión de audio Bluetooth.

Dispositivo de Audio: Este modo emula un dispositivo de audio haciendo que el sistema operativo pueda ver una conexión de audio Bluetooth como una tarjeta de sonido obteniendo ventajas de las facilidades de audio Bluetooth.

- *Ethernet*

El plugin Ethernet[30] emula una conexión Ethernet, habitando a aplicaciones estándares tales como el Explorador de Windows y NetMeeting para usar la conexión Ethernet Bluetooth como si estuvieran usando una red Ethernet real.

Se debe tener en cuenta que este módulo no envía y recibe los datos de la red directamente, sino que usa un driver especial para unirse con el sistema operativo permitiéndole al SO ver y usar la conexión Ethernet Bluetooth como una tarjeta de red normal.

- *OBEX Object Push*

El plugin Envío de Objetos OBEX envía y recibe datos a partir de un enlace Bluetooth establecido, para ésto ofrece una serie de funciones simples.

[30] Este modulo se basa en la información actual que presenta el grupo de trabajo Ethernet Bluetooth y como este no esta finalizado aún se considera propietario.

- Transferencia de Archivos OBEX

Este plugin le ofrece al programador un conjunto de funciones fáciles de usar que proporcionan acceso a la funcionalidad de transferencia de archivo OBEX.

- *Dial Up Networking (DUN)*

El plugin Dial Up Networking (Conexión por línea conmutada) sólo soporta el lado del cliente de una conexión DUN y por tanto se puede usar para establecer una conexión con cualquier dispositivo que implemente la funcionalidad del lado del servidor.

- *LAN*

El plugin LAN permite el acceso a una Red de Área Local usando el Protocolo Punto a punto (PPP), actualmente, este plugin sólo soporta el lado del cliente (Terminal de Datos) de una conexión LAN, por consiguiente, puede usarse para conectarse a cualquier dispositivo Bluetooth que implemente la funcionalidad en el lado del servidor.

La comunicación de los datos a través de la conexión LAN se soporta usando la ayuda incorporada dentro del sistema operativo Windows para la conexión de Cable Directa. Para configurar la conexión de Cable Directa se debe usar el puerto COM virtual Bluetooth que esta asociado con el perfil LAN, todos los datos dirigidos a través de la interfaz se envían sobre el enlace Bluetooth hacia el dispositivo remoto.

- *FAX*

Actualmente, este plugin sólo soporta el lado del cliente (Terminal de Datos) de una conexión FAX, por consiguiente, pueden usarse para conectarse a cualquier dispositivo Bluetooth que implemente la funcionalidad del lado servidor.

La comunicación de los datos a través de la conexión serial se soporta usando las aplicaciones Windows. Una conexión FAX Bluetooth se configura en Windows de la misma forma como una conexión de FAX se configura para un módem FAX normal.

- *Headset*

Este plugin se basa tanto en las conexiones seriales para señales de control de módem y comandos AT como en las conexiones para transferencia de audio, por lo tanto es dependiente de los módulos SDP, RFCOMM y Audio. Los dos dispositivos que toman la parte en una conexión Headset tienen fundamentalmente dos roles diferentes:

Gateway de Audio: Éste dispositivo actúa como maestro del audio, tanto para entrada como para salida. Dispositivos Gateway de Audio típicos son: teléfonos celulares y computadores personales.
Headset: Éste dispositivo actúa como mecanismo de entrada y salida remoto de las Gateway de Audio. Típicamente, los dispositivos Headset Bluetooth son auriculares especializados.

El soporte completo para ambos roles está incluido en la API:
Cuando se actúa como Gateway de audio, el plugin de Audio en el dispositivo local se fija en el modo Dispositivo de Audio. Cualquier audio hacia y desde la fuente (por ejemplo NetMeeting o un MP3 player) se dirige a la tarjeta de sonido virtual del dispositivo de audio Bluetooth. De esta manera el audio se enruta hacia el dispositivo Headset remoto en lugar de enrutarlo a una tarjeta de sonido real.

Cuando se actúa como Headset, el plugin de Audio en el dispositivo local se fija en el modo PC Speaker, y de esta manera un micrófono adjunto a la tarjeta de sonido del sistema puede usarse como dispositivo de audio de entrada, mientras los speakers adjuntos a la tarjeta de sonido del sistema actúan como dispositivo de salida.

- *IBluetooth*

Es un plugin utilizado para establecer una comunicación con el modulo Gestor de Enlace, como muchos de los comandos disponibles a través del Gestor de Enlace son muy complejos y ocasionalmente usados, este plugin proporciona un nivel de abstracción que no sólo oculta esos comandos, sino las interfaces para información especial, tales como las configuraciones de seguridad dependientes del dispositivo y la información almacenada de los dispositivos descubiertos.

- *Herramienta de configuración*

La Herramienta de Configuración es un Panel de control que ofrece una interfaz de usuario para gestionar los puertos COM virtuales de Bluetooth, esto incluye la creación o eliminación de los mismos además de la posible asociación de varios puertos COM a perfiles específicos. Estrictamente hablando, la herramienta de configuración realmente no hace parte del API, sin embargo, proporciona funcionalidades de configuración de las que dependen la mayoría de perfiles.

3.3.3 Como utilizar la API

Para dar una pequeña muestra de la utilización de esta API a continuación a manera de ejemplo se presentan los procedimientos y funciones que deben ser utilizadas para establecer una conexión de puerto serial y las tareas que se ven involucradas:

1. Establecer un enlace Bluetooth con el dispositivo remoto, teniendo en cuenta la configuración de seguridad establecida tanto por el dispositivo local como por el dispositivo remoto.
2. A partir del modulo SDP buscar los servicios de puerto serial disponibles sobre el dispositivo remoto.
3. Extraer de la información SDP de los dispositivos remotos información especifica como los atributos del servicio para que el usuario pueda realizar la mejor elección entre los múltiples servicios.
4. Establecer la conexión RFCOMM.

Sin embargo para llevar a cabo las anteriores tareas no se requiere interactuar directamente con los módulos mencionados, para esto el plugin Puerto Serial define dos funciones que permiten este procedimiento:

1. La función `GetRemoteServices()` se llama para buscar servicios de puerto serial en un dispositivo Bluetooth remoto, esta función requiere como parámetro de entrada las direcciones de los dispositivos Bluetooth y si se ejecuta correctamente retorna una lista de nombres de servicios que pueden ser desplegados en una interfaz de usuario. Un nombre de servicio para el perfil del Puerto Serial podría ser, por ejemplo, 'COM3'.
2. y posteriormente, se llama a la función `ConnectService()` para establecer la conexión. Esta función requiere como parámetros el nombre del servicio remoto y las direcciones tanto del dispositivo local como la del remoto.

Las funciones `IBTSerialPortProfile::GetRemoteServices` `IBTSerialPortProfile::ConnectService` son sincrónicas por lo tanto el resultado de las operaciones puede ser obtenido una vez la función retorne.

El ejemplo anterior ilustra algunos de los conceptos en que el API está basado:

- El uso de nombres de servicio "amigables" qué proporcionan una identificación única tanto a los servicios locales como a los remotos.

- Existe un plugin por cada perfil, y cualquier operación que involucre ese perfil puede ser obtenida usando ese plugin exclusivamente.
- El establecimiento de un enlace Bluetooth, manejo de roles maestro/esclavo, autenticación, encriptación son algunos de los procesos manejados automáticamente.
- No se tiene que establecer una comunicación directamente con el modulo SDP para solicitar o analizar información SDP para un perfil especifico ya que las tareas de descubrimiento de servicio locales y remotos se llevan a cabo automáticamente por los mismos plugins de perfiles.

3.3.4 Otro Ejemplo – Descubrimiento de un dispositivo y establecimiento del enlace

El siguiente ejemplo hace referencia a las diferentes funciones o métodos que deben utilizarse dentro de la definición del API para llevar a cabo las tareas correspondientes para el Descubrimiento de un dispositivo y el establecimiento de un enlace.

Indagación

1. Llamar la función `Inquiry()` que está definida dentro del módulo IBluetooth, esta función hace que el dispositivo entre en el estado Indagación para descubrir unidades Bluetooth que estén dentro del rango de cobertura. Las unidades que escuchen esta indagación contestarán con la dirección Bluetooth.
2. Para estar escuchando permanentemente los eventos que se generan como resultado de la indagación se debe implementar la interfaz `IBluetoothEvents`, por ejemplo cada vez que se encuentra un dispositivo un evento `InquiryResult` es reportado o cuando la indagación ya se ha realizado un evento `InquiryDone`.

3. Cuando la Indagación se completa se puede realizar un descubrimiento de nombre sobre los dispositivos. (solo uno al mismo tiempo).
   - El método `GetRemoteFriendlyName()` del modulo `IBLUETOOTH` se utiliza para preguntar por el nombre “amigable” del dispositivo remoto. Este método resulta del evento `RemoteFriendlyNameResult`.

Este es el algoritmo:

```
For    cada dispositivo encontrado
       llamar GetRemoteFriendlyName()
       esperar el evento RemoteFriendlyNameResult
       Extraer el nombre del evento
EndFor
```

3.4 COMPARACION ENTRE LA API JSR-82 CON RESPECTO A LAS OTRAS API'S

La tabla 3.1 hace una comparación de los aspectos más relevantes para la tecnología Bluetooth desde la perspectiva de cada API. Como se puede observar, las características que brindan las tres especificaciones de APIs son similares en cuanto a las capacidades, sin embargo existe una diferencia fundamental en cuanto al lenguaje de programación que cada una de ellas emplea y por ende la plataforma necesaria para su implementación y es por estas dos razones precisamente que se selecciona la Especificación JSR-82 como la herramienta mas adecuada para el diseño de un servicio telemático.

	Especificación JSR-82	**API C++**	**Especificación DIGIANSWER**
Soporte para Transmisión	Datos	Datos	Datos y Audio
Lenguajes de Programación	Java	C++	C++, Delphi y Visual Basic
Sistemas Operativos	Cualquiera	Symbian	Windows
Capacidades	• Registro de Servicios. • Descubrimiento de dispositivos y servicios. • Establecimiento de conexiones L2CAP y OBEX. • Gestor de Seguridad	• Registro de Servicios • Descubrimiento de dispositivos y servicios. • Establecimiento de conexiones L2CAP y RFCOMM. • Configuración de requerimientos de seguridad.	• Registro de Servicios. • Descubrimiento de dispositivos y servicios. • Establecimiento de conexiones L2CAP, RFCOMM y OBEX. • Configuración de Seguridad.
Protocolos del Stack de Bluetooth	• Protocolo de Adaptación y Control del enlace Lógico (L2CAP) • Protocolo de Descubrimiento de Servicio (SDP) y OBEX	• L2CAP • SDP • RFCOMM	• L2CAP • SDP • RFCOMM • OBEX
Perfiles Bluetooth	Perfil de Acceso Genérico (GAP), Perfil de Aplicación de Descubrimiento de Servicios (SDAP), Perfil de Puerto Serial (SPP), (Perfil de Intercambio de Objetos genérico (GOEP).	No hace referencia a perfiles	• SPP • GOEP • LAP • SDAP • FP • HS • FTP

Tabla 3.1 Comparación entre APIs

Java es un lenguaje de desarrollo, orientado a objetos, de propósito general desarrollado por *Sun Microsystems*, y como tal es válido para realizar todo tipo de aplicaciones. Incluye una combinación de características que lo hacen único y está siendo adoptado por multitud de fabricantes como herramienta básica para el desarrollo de aplicaciones comerciales de gran repercusión. Los

grandes logros que ha alcanzado la plataforma Java se basan en el hecho de ser simultáneamente una tecnología multiplataforma e independiente de otra plataforma.

Todo lo anterior resulta posible porque la tecnología Java consiste en dos componentes básicos: en el lenguaje de programación propiamente dicho (que permite crear aplicaciones para fines específicos) y en un «ambiente» sobre el cual estas corren.

Para ejecutar una aplicación Java, en cualquier sistema operativo (ya sea Windows, Linux, Unix, Solaris, Mac OS, Symbian. etc.) el usuario debe tener preinstalada una máquina virtual java. Esta integra, hoy en día, la generalidad de los sistemas operativos para ordenador y PDA's, y también (de forma creciente) en los teléfonos móviles.

La verdadera importancia radica en el hecho de que al realizar en Java sus aplicaciones, el programador tiene la certeza de que ellas son universalmente compatibles y funcionarán en cualquier sistema operativo. Java constituye así una especie de lenguaje universal, un puente entre sistemas operativos.

Los dispositivos portátiles sean agendas electrónicas (PDA's, Palm Pilots o iPaqs, como se conocen comercialmente) o teléfonos móviles presentan limitaciones de algunos recursos a nivel del hardware (memoria, capacidad de almacenamiento y procesamiento. etc.) sustancialmente inferiores a los de un computador, partiendo de esto existen fundamentalmente tres versiones de la plataforma Java en función del propósito que se sigue y el medio en el que va a funcionar: el Java Enterprise Edition (J2EE), concebida para empresas y servidores con elevada exigencia; el Java2Standard Edition (J2SE), implementada para uso doméstico e individual y el Java 2 Micro Edition (J2ME), concebida para pequeños dispositivos de masa, a nivel de la electrónica de consumo, como por ejemplo los teléfonos.

La justificación es fácil de entender. Basta pensar en un teléfono que tiene 4 MB de memoria dinámica, sensiblemente la mitad del tamaño de la MVJ usada en un ordenador medio. No solo no sería posible acomodar la Maquina Virtual de la Edición Estándar dentro de los recursos del teléfono como sería inútil que ésta soportara muchas funcionalidades sin uso concreto en aplicaciones concebidas para un teléfono, sin embargo, a pesar de funcionar sobre una plataforma en común las aplicaciones Java obedecen, según el medio donde deban funcionar, a especificaciones particulares. En el caso de la J2ME podemos, simplificadamente, verla como una versión «abreviada y adaptada» del Java que equipa a dispositivos más complejos.

Algunos estudios estiman que en el año 2002 cerca del 11% de los teléfonos móviles vendidos disponían de tecnología Java y que, a partir de 2005, la mayoría de los terminales vendidos estarán equipados con ella. Si se quiere hacer de un teléfono algo más que un dispositivo para hacer y recibir llamadas, tocar melodías y exhibir logotipos; convertirlo en un instrumento dinámico capaz de aceptar todas las implicaciones que el usuario quiera (desde juegos a programas para calcular mareas; planetarios; hojas de cálculo, etc.) valdrá la pena invertir en un terminal dotado con tecnología Java, además según muchos expertos en el sector de las telecomunicaciones, el uso combinado de Java y Bluetooth será de gran utilidad para algunos tipos de sistemas empotrados, proporcionando un entorno multiplataforma y de servicios distribuidos.

Beneficios del JS-R82 para Bluetooth

El propósito de la especificación JSR-82 es estandarizar un conjunto de APIs que le permitan integrar los dispositivos habilitados con la tecnología Java en un ambiente Bluetooth. Entre los beneficios más importantes que se tienen a la hora de implementar el JSR-82, están:

- Potabilidad del Código. La API estándar permite que el mismo código funcione en dispositivos diferentes y aun en Pilas de protocolos de Bluetooth diferentes.
- Aumenta la Adopción de Bluetooth puesto que reduce el tiempo de desarrollo para las aplicaciones. Esto se explica porque actualmente se cuenta con el respaldo de una comunidad de desarrolladores que esta creciendo rápidamente.
- El entorno de desarrollo de las aplicaciones Bluetooth es flexible y abierto - Java ofrece un estándar no propietario para desarrollar aplicaciones que permitirán el éxito de la tecnología inalámbrica Bluetooth.
- El software portátil para Bluetooth aumenta el tiempo de comercialización - Java permite la reutilización del software en todos los dispositivos habilitados para el JSR-82.
- El funcionamiento es independiente de la interfaz de radio y de la pila de protocolos de Bluetooth.

CAPÍTULO 4 DESCRIPCIÓN DEL PROTOTIPO DE SERVICIO TELEMÁTICO Y DEFINICIÓN DE LA ARQUITECTURA

Con el propósito de complementar el estudio detallado que se realizó en el capitulo 2 acerca de los perfiles Bluetooth para desarrollar una aplicación e implementar la mejor herramienta o API que cumpla con las expectativas que esta tecnología inalámbrica genera, se realizó el diseño de un prototipo de servicio telemático para Bluetooth basado en la plataforma java. Teniendo en cuenta que las aplicaciones desarrolladas con Bluetooth se pueden enmarcar, como se vio en capítulos anteriores, en modelos de uso que pretenden dar respuesta a necesidades de comunicación de dispositivos móviles, en este proyecto en particular se diseñó el servicio Punto de Acceso Inalámbrico a una LAN, pretendiendo con el mismo constituir la base conceptual y practica para el diseño e implementación de nuevos servicios telemáticos.

4.1 Descripción del Prototipo del Servicio

El servicio cumple con la funcionalidad primordial de permitir a un usuario de un dispositivo móvil establecer una conexión inalámbrica con una LAN y posteriormente a través de un servidor de autenticación validarse para obtener los permisos suficientes y así poder hacer uso de los servicios que presta dicha LAN entre los cuales se puede encontrar servicios de transferencia de archivos, servicios de impresión, de Internet entre otros..

Enseguida se hace una pequeña descripción del funcionamiento básico del servicio:

- El servicio se inicia cuando el usuario del dispositivo móvil una vez activa la aplicación selecciona la opción que le permite conectarse a la LAN, en ese mismo instante la aplicación del cliente responde con una búsqueda de dispositivos cercanos que presten el servicio de acceso (servidor de acceso); una vez encontrado el dispositivo que permite ingresar a la LAN, el usuario establece una conexión con el servidor; posteriormente el usuario debe identificarse para verificar cuales son sus permisos dentro de la red y poder utilizar los recursos y servicios que ofrece la misma.

- El servidor de autenticación es el encargado de validar los datos ingresados por el usuario y autorizar a este para que utilice determinados servicios que presta la red, cuando este proceso finaliza se informa lo ocurrido al usuario.
- Cuando por algún inconveniente en el enlace se cancela el proceso de conexión, se notifica al usuario, con lo cual concluye la operación; el usuario también puede cancelar la conexión.

Teniendo en cuenta lo anterior y con el objeto de realizar una descripción mas completa del servicio, a continuación se presenta el resumen y descripción de los Casos de Uso Esenciales de la Aplicación:

Caso de Uso: Establecer Conexión

Propósito: Establecer la conexión física entre dos dispositivos de manera que estos puedan reconocerse y hacer uso de los servicios que ofrecen. Registrar servicios para que otros dispositivos puedan encontrarlos y registrarlos para si mismos y acceder a ellos.

Resumen: El usuario inicia la aplicación (se establecen parámetros especiales para llevar a acabo la conexión de dispositivos) y procede a ejecutar una búsqueda de dispositivos en un entorno próximo. En caso de que se encuentre uno o más dispositivos, la aplicación indica al usuario cual es el nombre y la dirección del dispositivo que se descubrió. Una vez encontrado el dispositivo, la aplicación inicia la búsqueda del servicio de acceso a la LAN. Si el dispositivo remoto ha registrado este servicio con anterioridad, este podrá descubrirse y el usuario podrá registrarlo localmente para poder conectarse con él. Si la conexión a la LAN es exitosa, el dispositivo remoto responderá afirmativamente. A partir de este momento el usuario puede seguir utilizando los servicios que ofrece la red como tal o presionar el botón de desconexión que le permite desregistrar el servicio de la base de datos local y activar nuevamente la aplicación.

Caso de Uso: Validar Acceso

Propósito: Validar al usuario que ya está conectado a la red para que pueda hacer uso de determinados servicios que se prestan.

Resumen: Una vez el usuario ha sido conectado a la LAN puede utilizar cualquier servicio que esta preste, cuando él hace la solicitud aparece un cuadro de dialogo donde se le solicita al usuario ingrese un nombre de usuario y la respectiva contraseña. Estos datos son enviados al servidor de

autenticación el cual es el encargado de verificar dichos datos y autorizar al usuario para el uso de determinados servicios. Finalmente el servidor de autenticación informa al usuario los resultados obtenidos.

Para mayor detalle de la descripción del servicio construido se puede consultar el *Anexo B Modelado de la Aplicación* donde se presenta los productos generados para su desarrollo.

4.2 Definición de la Arquitectura

Los componentes que hacen parte del servicio telemático y la interacción que existe entre ellos conforman la arquitectura final del mismo, como se puede observar en la Figura 4.1

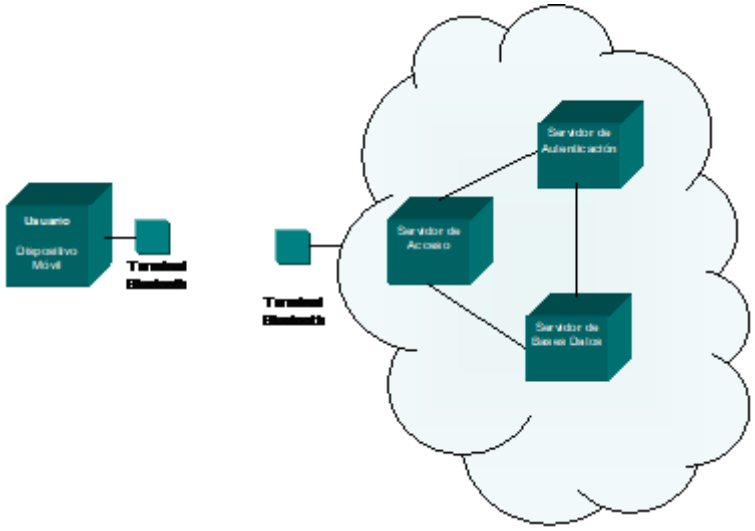


Figura 4.1 Componentes del Servicio

Como se puede apreciar el servicio telemático esta soportado por una aplicación cliente que se encuentra alojada en el dispositivo móvil del usuario; una aplicación servidor de acceso, un servidor de bases de datos y un servidor de autenticación instalados en equipos que hacen parte de una Red de Área Local LAN con características especificas que constituyen los requisitos funcionales esenciales para el desempeño del servicio como tal.

Descripción de los componentes

Dispositivo Móvil:

Este módulo representa al dispositivo móvil ya sea un portátil, PDA, teléfono móvil o cualquier dispositivo personal donde la aplicación del cliente está alojada, su principal funcionalidad es recibir las peticiones del usuario y realizar las respectivas solicitudes al servidor. Este módulo a través de la interfaz serial se comunica con el terminal Bluetooth del lado del cliente.

Terminal Bluetooth:

Este dispositivo corresponde al Hardware Bluetooth, el cual por medio de software proporcionado por las aplicaciones tanto del lado del cliente como del lado del servidor permiten soportar la comunicación de datos o audio que se establece entre el Dispositivo Móvil y el Servidor de acceso. En general es el dispositivo que soporta la conexión inalámbrica y brinda la interfaz de radio que la tecnología Bluetooth requiere.

Como se mencionó en el Capitulo 2 acerca del Perfil de Puerto Serial, este se encarga de definir los requerimientos necesarios para que los dispositivos Bluetooth establecieran las conexiones emuladas de cable serial, a través del nivel RFCOMM, entre los dos dispositivos finales. (Dispositivo Móvil y Servidor de Acceso).

Servidor de Acceso:

El módulo correspondiente al servidor de acceso define el servicio de acceso a una LAN y proporciona este servicio a los clientes remotos almacenándolo dentro de un registro de servicio asignado para cada dispositivo en la base de datos del dispositivo local (servidor). Posteriormente, esta aplicación del servidor espera por una solicitud del cliente que recibe a través del terminal Bluetooth del lado del servidor con el cual mantiene una comunicación a través de la interfaz serial para iniciar el contacto con el servidor de acceso.

Este módulo se encuentra alojado en un equipo que hace parte de la LAN.

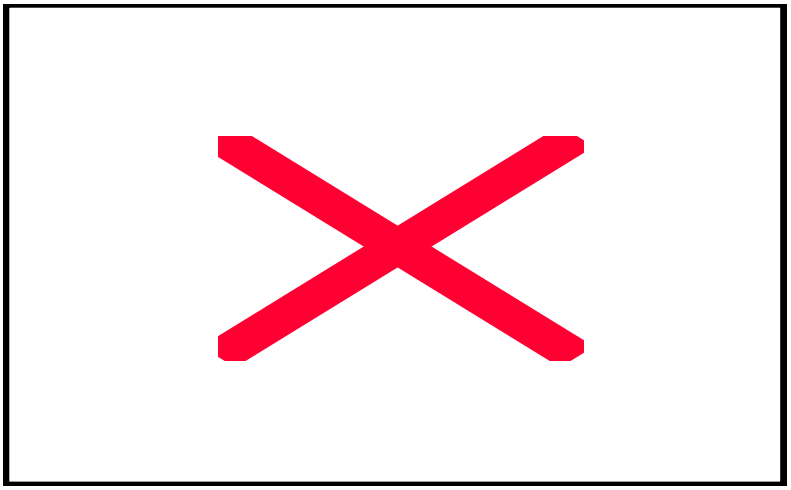
Servidor de Autenticación:

Al igual que el módulo anterior, éste módulo reside en un equipo que hace parte de la LAN, constantemente escucha peticiones de los clientes una vez han establecido la conexión con el servidor de acceso y deseen utilizar los servicios ofrecidos. Este servidor realiza una conexión con el servidor de Bases de Datos para verificar la validez de los datos suministrados por el usuario.

Servidor de Bases de Datos:

Finalmente se tiene el modulo servidor de bases de datos que es el encargado del almacenamiento de toda la información necesaria que maneja tanto el servidor de acceso como el de autenticación. Este modulo también esta instalado dentro de un equipo que pertenece a la LAN.

Y finalmente se muestra la arquitectura de los servicios diseñados:



En la parte del servidor tenemos la aplicación del lado del servidor que va a encargarse de la publicación de los servicios que se van a prestar a los dispositivos remotos y ademas preparar a cada uno de ellos para que espere por las peticiones de los clientes. Cada uno de los servicios es manejado como un hilo independiente del otro y por lo tanto tiene su propia lógica de servicio.

En la parte del cliente tenemos la aplicación del lado del cliente que es la que realiza las transacciones de búsquedas de dispositivos y servicios; una vez se tiene el registro del servicio que se va a utilizar se crea un hilo correspondiente a este servicio del lado cliente que cumple con los requisitos que exige el servidor

Y finalmente la comunicación que se establece entre el cliente y el servidor a partir del servicio se realiza a través del Staff de protocolos de Bluetooth.

CONCLUSIONES Y RECOMENDACIONES

La tecnología Bluetooth se plantea como candidata a convertirse en la tecnología de transmisión vía radio de corto alcance en múltiples aplicaciones y entornos para los que actualmente se recurre a conexiones físicas por cable o bien a transmisión por infrarrojos. Además, Bluetooth es un sistema suficientemente flexible para ofrecer la posibilidad de desarrollar aplicaciones o productos a medida que se requieran de las facilidades de transmisión vía radio proporcionadas por la tecnología.

Bluetooth está pasando por las etapas tradicionales de pruebas, conflictos y correcciones. Una importante ventaja en el caso de Bluetooth, es la fuerza de más de 2000 empresas que integran el SIG, las cuales dan aportes importantes tanto en el campo técnico, como en los campos de mercadeo y publicidad. El haber tomado en cuenta los problemas de otras tecnologías similares, al momento de planificar el desarrollo del estándar, ha sido muy provechoso, ya que se han logrado disminuir las dificultades habituales. El estándar Bluetooth tiene como soporte una fuerte maquinaria creativa y una clara visión de futuro, requerimientos mínimos de todo movimiento exitoso. Aunque siempre hay puntos dispares en el camino, como los problemas de interferencia con otras tecnologías, siempre ha sido posible encontrar acuerdos de cooperación. Para el nicho de mercado que persigue la tecnología Bluetooth se cuenta con una alta posibilidad que en algunos años se incorpore transparentemente a nuestra vida diaria, tal como lo hicieron el teléfono, la televisión, el computador e Internet.

Actualmente el mayor enfoque del trabajo está en el desarrollo de un estándar que permita garantizar la interoperabilidad de los dispositivos Bluetooth elaborados por diversos fabricantes. Existen todavía algunos problemas a solucionar, como el precio de la tecnología y la posibilidad de controlar todas las funciones desde un solo dispositivo, no obstante con las empresas envueltas en el proyecto, no se puede dudar del suceso. Por las facilidades que ofrece la tecnología Bluetooth, los dispositivos provistos de interfaz Bluetooth serán tremendamente útiles… para aquel que lo necesita. Todos los productos y estándares tienen pros y contras; el éxito lo van a obtener quienes manejen la tecnología adecuada, pero será más significativo aún entender mejor el negocio y las necesidades presentes y futuras de los usuarios, a los cuales se les debe brindar una herramienta útil que pueda ayudarlo en su vida cotidiana.

El escenario para Colombia no es muy diferente del resto del mundo. En el país se está viviendo una época de transición tecnológica, modernizando su infraestructura de comunicaciones y masificando poco a poco el acceso a la misma. Casos como el de la telefonía móvil de segunda y tercera generación (PCS y otras que usan GSM), implican más y mejores servicios (transmisión de audio y video con buena definición), que promueven e incentivan el uso de tecnologías como Bluetooth.

Además del continuo desarrollo en cuanto a nuevos perfiles y escenarios, el Bluetooth SIG avanza en la conformacion de otros frentes para mejorar la especificación, dichos frentes incluyen asuntos relacionados con la interoperabilidad. En abril de 2002, Microsoft liberó el primer ratón y teclado inalámbrico que se comunica con un computador personal a traves de la tecnología Bluetooth. La ratificación de Bluetooth por la IEEE en el mes de marzo de 2002 es un triunfo importante para el Bluetooth SIG, el cual ha luchado por mucho tiempo por ser un protagonista de la sopa de letras de los estándares de redes inalámbricas en el mundo. Esta ratificación le da a Bluetooth más credibilidad y permitirá más desarrollos en la industria encaminados a satisfacer necesidades más específicas de los usuarios finales a un bajo costo y buen desempeño.

Bluetooth será una de las tecnologías mas importantes, más de lo que ya es, por otro lado es una oportunidad, pues aun no se han agotado las áreas de trabajo. De cualquier manera es bueno mencionar que durante esta investigación también se encontraron opiniones encontradas acerca de la tecnología, muchas de ellas enfocaban a Bluetooth como una tecnología sin futuro o que en este momento no parecía un buen negocio… habrá que esperar a ver a quien le da la razón el tiempo.

5.1 Aplicaciones para Bluetooth

El futuro de Bluetooth está lleno de expectativas tanto para las mas de 2000 compañías pertenecientes al Bluetooth SIG como para los usuarios de dispositivos habilitados para la tecnología Bluetooth. Cada desarrollador ha colaborado en la visión conjunta que se tiene para la tecnología, y los casos de uso que están planeados son altamente diversos.

Por el lado de la IEEE, se espera que Bluetooth conforme la norma 802.15.2 de coexistencia con las redes WLAN y que surjan versiones de alta y baja velocidad, para aplicaciones de multimedia y de dispositivos de baja complejidad respectivamente. Al crearse estos estándares, se ampliarían aún más las posibilidades para el uso de Bluetooth, por ejemplo, para el modelo de baja velocidad y baja complejidad se esperan las siguientes aplicaciones:

- Sensores

- Juguetes interactivos
- Carnets inteligentes
- Controles remotos
- Dispositivos para la automatización del hogar

En general, dispositivos que deben tener un alto grado de simplicidad, bajo costo, bajos requerimientos de tasa de transferencia y que deben mantener una vida de batería de varios meses o varios años.

Por el lado del Bluetooth SIG y sus compañías patrocinadoras se ha creado ya una buena cantidad de soluciones propuestas y además se han implementado una considerable cantidad de escenarios de prueba, algunos de los más interesantes se explicaran a continuación:

- Automatización Hotelera: la cadena de hoteles Holiday Inn, Axis Communications AB y Registry Magic han unido sus esfuerzos para convertir el Holiday Inn de Wall Street en el primer hotel en prestar servicios inalámbricos. La tecnología permitirá a los huéspedes hacer su "check-in" y "check-out", entrar a sus habitaciones, utilizar Internet, recibir mensajes de voz, y pagar comidas en el restaurante del hotel. Paralelamente, hay iniciativas como la de la compañía i-Wap para simplificar tareas de la administración hotelera, como por ejemplo el monitoreo de consumo del mini-bar, y el manejo de órdenes de "Room-Service". Todavía está por verse el impacto que pueda tener Bluetooth sobre la operación hotelera y si podrá influir positivamente en la captación de nuevos clientes. Sin embargo se trata de una tecnología que muy fácilmente puede significar un punto clave en la historia de esta industria si se juegan correctamente las cartas.

- Acceso a Internet en aeropuertos: muchos ejecutivos que viajan regularmente se ven cada vez más en la necesidad de tener acceso contínuo a su correo electrónico y a la Web. Hasta ahora la solución adoptada por muchos suele ser una llamada telefónica a un número de acceso de su proveedor o una complicada y costosa conexión a través de un teléfono celular. Actualmente, compañías como American Airlines y TWA están implementando el servicio de Internet Inalámbrico en sus salas de espera de primera clase. A pesar de los esfuerzos de ambas compañías, han encontrado un gran número de obstáculos para la implementación del servicio. El hecho que principalmente ha retardado la implementación de Bluetooth dentro de ésta área es que se trata de una tecnología en vía de desarrollo. Adicionalmente las operadoras de los aeropuertos y las compañías de

telefonía de larga distancia, al ver sus intereses en peligro, han comenzado a dificultar deliberadamente la entrada de las WPANs en las instalaciones de los aeropuertos.

- Acceso a información en trenes: BT Syncordia y Midland Mainline han comenzado pruebas para verificar la factibilidad de utilizar intranets basadas en Bluetooth dentro de sus vagones. Dentro de los trenes, los usuarios tendrian acceso a ciertas páginas Web de noticias obtenidas con antelación y almacenadas en un "Cache" del tren. Posiblemente se contempla la posibilidad de que los pasajeros adquieran sus tickets a través de este medio. Una conexión "en vivo" en los trenes haría mucho más interesante la implementación de ésta propuesta.

5.2 Inconvenientes de la Tecnología Bluetooth

Algunos de los obstáculos que presenta la tecnología Bluetooth están relacionados con las problemáticas noticias acerca de su seguridad y fallos imprevistos. Los más críticos dicen que Bluetooth podría sufrir por una propaganda desorbitada y por la falta de prueba, pero los analistas aún están convencidos de que la tecnología prevalecerá. Además para los criticos de Bluetooth existen ciertos incovenientes que hacen poco viable el crecimiento del estandar, entre los que se encuentran los siguientes:

- Bluetooth podria caer en la misma trampa que han experimentado los servicios de WAP y por ende una pobre reacción del consumidor, debido a que los primeros productos Bluetooth fueron diseñados siguiendo la misma linea que presentaban los servicios para WAP.
- El proceso de prueba comenzó desde diciembre de 1994 pero aún falta parte del modelado puesto que no todos los servicios están disponibles en este momento. Es por ello que resulta imposible asegurar que un servicio funcione con otro hasta que no existan ambos.
- Costará algún tiempo solucionar estos problemas debido a que los vendedores de las aplicaciones deben tener acceso al hardware Bluetooth antes de que se haya comenzado a probar el software.
- Hay también problemas de interferencias. Bluetooth funciona en la misma radio frecuencia que el estándar inalámbrico LAN 802.11, así que las interferencias con LANs es un problema que aún está por resolver. Hay asuntos de seguridad también. La industria aérea desconfía de la habilidad de Bluetooth para responder automáticamente cuando éste reciba una señal, lo cual supondría un gran problema durante el vuelo.
- Los miembros de la industria de la informática están preocupados por las transacciones inalámbricas. Bluetooth proporciona un nivel de seguridad de enlace pero no mucho más.

Los fabricantes de teléfonos móviles están trabajando sobre la marcha para unir el protocolo de transmisión de datos de Internet con Bluetooth, pero aun no hay algo oficial.

5.3 Puntos Fuertes de la Tecnología Bluetooth

Sin duda Bluetooth marca el inicio de una nueva filosofía basada en la manera cómo interactuan diferentes dispositivos electrónicos, aunque existen otras tecnología rivales (IrDA: conexión mediane infrarrojos, IEEE 802.11, UWB: Ultra Wide-Band Radio, Home RF...) sin duda alguna Bluetooth lidera esta serie de tecnologías gracias a varios puntos fuertes, entre los que se destacan los siguientes:

- Requisitos de hardware mínimos.
- El precio de los componentes Bluetooth es muy bajo.
- El consumo de los componentes Bluetooth es también muy reducido.

Todas estas ventajas permiten incorporar la tecnología Bluetooth en dispositivos electrónicos a un precio muy bajo y confiere a esta tecnología una posición única de mercado, y ahora que ya han sido introducidos en el mercado muchos modelos de chips Bluetooth y una vez desarrollados y distribuidos diferentes SDKs (Software Development Kits) se espera que esta tecnología comience a afianzarse con mucha fuerza. Prueba de ello es la llegada de la implementación de Bluetooth para los sistemas operativos Windows XP y Macintosh OS X 10.2 – los cuales si funcionan adecuadamente - probablemente ayudarán a la adopción de Bluetooth.

5.3.1 Prevención frente a las interferencias

El amplio potencial del mercado para las soluciones Bluetooth parecen ser los teléfonos celulares. Las raíces del Bluetooth están en el GSM y las previsiones son de un rápido crecimiento del Bluetooth en los mercados GSM. Sin embargo, poner un potente radio celular próximo a un radio Bluetooth de baja potencia en un teléfono celular requiere un cuidadoso diseño por la posibilidad de transmitir y recibir interferencias entre las dos radios.

El mayor problema para añadir Bluetooth a los teléfonos celulares es el potencial del poderoso transmisor del teléfono celular que bloquea al receptor de Bluetooth durante la transmisión, este problema afecta a los sistemas celulares half-duplex (la radio celular o transmite o recibe, pero no hace ambas simultáneamente) tales como el TDMA basado en el estándar GSM, así como los sistemas dúplex completos (la radio celular puede transmitir recibir simultáneamente) tales como CDMA o FM análogo. Estas interferencias de teléfono celular están causadas por ruido en la banda ancha generado por el transmisor celular. Básicamente el teléfono celular transmitirá no sólo la señal de datos requerida sino también cierto nivel de ruido. Algo de este ruido aparecerá en la

banda de Bluetooth. El nivel de este ruido puede ser suficiente para interferir o bloquear una señal que llega de Bluetooth. Por lo tanto al desarrollar sistemas Bluetooth se usan filtros de radio especiales que pueden funcionar a pesar de las interferencias internas de la señal de radio del GSM. El ruido de la banda ancha es normalmente producido por la señal transmitida que se encuentra más allá de los límites legales de transmisión. En los tres tipos de sistemas GSM, por ejemplo, el ruido de banda ancha que cae en ciertas bandas está restringido.

Para asegurarse de que el módulo de radio de Bluetooth opera efectivamente dentro de un teléfono celular, el nivel de ruido del teléfono transmisor es medido y controlado y esto se implementa relativamente fácil en los diseños de los teléfonos de hoy.

El segundo problema en la aplicación del teléfono celular es el ruido del transmisor Bluetooth que bloquea el receptor del teléfono celular, el deseo es mantener el ruido transmitido por el transmisor Bluetooth del receptor del teléfono celular menor o igual al ruido del canal.

Cada estándar celular presenta retos específicos para Bluetooth. Por encima de todo Bluetooth tiene una innovadora y bien pensada arquitectura para sobrevivir en este complicado entorno de radio, pero solo a traves de extensos tests de radio se consiguen asegurar soluciones de alta realización.

Podemos concluir que Bluetooth puede considerarse como una tecnología para redes inalámbricas pequeñas y seguras que puede proveer a los usuarios de conectividad transparente con otros dispositivos también habilitados. Debido al renombre e influencia que tienen las empresas involucradas en el desarrollo y mejoramiento de este estándar, es de esperarse que pronto la mayoría de los dispositivos personales sean desde fábrica "bluetooth enabled", con lo que casi sin darnos cuenta todos seremos partícipes de un gran salto en la aplicación de la tecnología inalámbrica en la vida diaria.

Para un futuro cercano el Bluetooth SIG ya está trabajando en la versión Bluetooth 1.2, la cual ofrecerá anchos de banda desde 2 Mbps a 3 Mbps. Y para un futuro no tan cercano también están trabajando ya en Bluetooth 2.0, el cual ofrecerá conexiones entre 4,8 y 12 Mbps.

Los nuevos Chips que usará Bluetooth 2.0 serán un 20% más caros que los actuales. En la nueva versión también se mejorará el servicio poniendo a todos los usuarios en el mismo nivel de tal

manera que no exista alguien superior que cuando abandone la conexión tire a todos los usuarios que están detrás de él, y que es la manera que tiene Bluetooth 1.1 de trabajar.

Bluetooth se creó principalmente para conectar pequeños dispositivos entre sí, pero actualmente las compañías están desarrollando software para usar Bluetooth para conectarse a redes wireless y acceder a internet.

5.4 Observaciones y Recomendaciones

1. Las APIs deberian implementar todos y cada uno de los protocolos del estandar Bluetooth, el principal inconveniente para la ejecución de algunos servicios radica en la falta de implementacion de algunas capas que están definidas dentro del Stack de Protocolos de esta tecnología y que resulta obligatoriamente necesarios para su funcionamiento.
2. Es importante tener en cuenta que a la hora de implementar nuevas y mejores soluciones software, tambien se considere la posibilidad de construir una herramienta hardware para Bluetooth con lo que se consigue mas flexibilidad a la hora de implementar los servicios, puesto que cada kit de desarrollo disponible en el mercado trae consigo una verticalidad en el software que implementa lo que se traduce en un limitado acceso a las funcionalidades de las capas mas bajas de Bluetooth.
3. Aunque resulta un trabajo extenso y complejo, se deberia estudiar la posibilidad de implementar una API que permita complementar el trabajo del estandar y que contribuya con el proceso de Interoperabilidad que se adelanta por parte del SIG.
4. Este trabajo se puede complementar con el estudio tanto de los nuevos perfiles que en este momento están siendo estudiados como de los nuevos modelos de uso que se generan a partir de éstos, en la versión Bluetooth 2.0

GLOSARIO

ACL: Asynchronous Connectionless (enlace asíncrono no orientado a la conexión).

API: application program interface (Interfaz del programa de aplicación).

Asimétrico: Describe un flujo de datos de ida y vuelta que van a velocidad diferente en cada dirección. Por ejemplo, en el DSL (Bucle Digital de Abonado) Asimétrico, los datos van de Internet hacia el abonado a una velocidad mucho mayor que en sentido contrario.

Asíncrono: Literalmente, no sincronizado. Se suele utilizar para referirse a comunicaciones en las que el flujo de datos no va unido a una señal específica de reloj - los módems transfieren datos de modo asíncrono.

ATM: (modo de transferencia asíncrona) dentro del medio informático hace referencia a la conmutación de paquetes (cells -- celdas o células) de un tamaño fijo con alta carga, rápida velocidad (entre 1,544 Mbps. y 1,2 Gbps) y una asignación dinámica de ancho de banda. También se conoce como "paquete veloz" (fast packet).

Auriculares y micrófono inalámbricos: Conjunto de auriculares y micrófono que se conectan sin cables.

Autenticación: Mecanismo de seguridad que impide el acceso a datos sensibles y hace imposible falsificar el origen de un mensaje.

Banda base: Ancho de banda reducido.

Banda ancha: Ancho de banda grande.

Banda ISM: Banda Industrial, Científica y Médica, un conjunto de frecuencias radio centradas en los 2.4 GHz que son de uso público libre en todo el mundo por dispositivos inalámbricos como los de Bluetooth.

BD_ADDR: dirección del dispositivo Bluetooth.

Calidad de Servicio: Usada en el contexto de las comunicaciones de datos, se refiere a la capacidad de garantizar cierto nivel de servicio para un tipo de tráfico en particular, casi siempre voz o vídeo, que son muy sensibles - aunque de maneras distintas - a las pérdidas de datos o desfases en la llegada.

Capa MAC: La capa de Control de Acceso a Medios de una red – en otras palabras, la circuitería - habitualmente una tarjeta de acceso a red - que gestiona el acceso a la capa física de la red - típicamente el cable o fibra, o también un enlace inalámbrico.

CardBus: Una interfaz a 32 bits que describe cómo se comunican las PC Cards con un ordenador, habitualmente un notebook.

CDMA: Acceso Múltiple por División de Código. Es un estándar de transmisión que separa canales de voz usando tecnología de espectro ensanchado.

Cifrado: Mecanismo de seguridad que evita las escuchas, manteniendo el privado el acceso a la comunicación.

Conmutación de circuitos: Describe un circuito de comunicación que se mantiene disponible para los que se comunican todo el tiempo que dura la comunicación - este es efectivamente el modo en que se realiza una llamada telefónica estándar. Tiene la ventaja de asegurar una capacidad de flujo constante, pero es muy poco eficiente en el uso del ancho de banda debido a los silencios que hay entre frases o incluso dentro de una frase o una palabra.

Conmutación de paquetes: Describe una señal de datos que ha sido partida en trozos o paquetes. Estos trozos pueden enviarse individualmente a sus destinos como si fuesen paquetes reales a través del sistema de conmutación, con la ventaja de un mensaje o una serie de mensajes no se apoderan de un canal de comunicación, permitiendo una eficiencia mucho mayor en el uso del ancho de banda disponible. Ver conmutación de circuitos.

DECT: Sistema de señalización con el que funcionan los teléfonos inalámbricos digitales.

Driver: controlador que permite gestionar los periféricos que están conectados al ordenador.

DSP: Procesador Digital de Señal - un chip diseñado específicamente y optimizado para el procesamiento de señales como voz y vídeo.

EPOC: Sistema operativo a 32 bits usado en los ordenadores de mano Psion Serie 5, que incluye una serie de aplicaciones, interfaces de usuario personalizables, opciones de conectividad y varias herramientas de desarrollo.

Esclavo: Dispositivo en una picored controlado por otro dispositivo (el maestro).

Espectro ensanchado: Se refiere al tipo de salto de frecuencia usado por Bluetooth, que salta, una tras otra, a través de un espectro de frecuencias de radio. Otra técnica, llamada espectro ensanchado de secuencia directa, reparte paquetes de datos entre varias frecuencias al mismo tiempo, pero es menos económico, consume más energía y no permite la coexistencia de varios puntos de acceso transmitiendo y recibiendo señales en la misma área porque bloquearían mutuamente su transmisión.

FireWire: Nombre dado por Apple al estándar IEEE 1394.

FIRMWARE: parte del software de un ordenador que no puede modificarse por encontrarse en la ROM o memoria de sólo lectura, Read Only Memory. Es una mezcla o híbrido entre el hardware y el software, es decir tiene parte física y una parte de programación consistente en programas internos implementados en memorias no volátiles. Un ejemplo típico de Firmware lo constituye la BIOS.

GAP: generic access profile (perfil de acceso genérico).

GOEP: generic object exchange profile (perfil genérico de intercambio de objetos).

GPRS: Servicio General de Paquetes Radio - una tecnología que envía datos en paquetes sobre las transmisiones GSM con el objetivo de aumentar el flujo de datos compartiendo ancho de banda, lo que permite a los usuarios alcanzar velocidades de transmisión de hasta 115 Kbit/seg.

GSM: Sistema Global de comunicaciones Móviles, la tecnología estándar de telefonía celular usada en Europa y la mayor parte de Asia y África.

HCI: host controller interface (interfaz controladora del host).

HEC: chequeo de redundancia cíclica de encabezados.

IrDA: Siglas de la Asociación de Datos por Infrarrojos, grupo de empresas cuyo nombre se ha convertido en sinónimo del estándar que desarrollaron para comunicaciones basadas en infrarrojos, que ha sido adoptado plenamente en la industria informática pero no en dispositivos de consumo como los mandos a distancia.

Java: Tecnología creada por Sun Microsystems que permite ejecutar sus programas sobre casi cualquier tipo de ordenador que incluya una Máquina Virtual Java (JVM), con el resultado de que Java es casi totalmente independiente de la plataforma. Estos programas suelen ser pequeños, para que puedan descargarse rápidamente a través de una conexión a Internet para ser ejecutados por el navegador.

LAN inalámbrica: Una Red de Área Local cuya capa física - el equivalente al cableado - se ha sustituido por ondas de radio.

L2CAP: logical link controller and adaptation protocol (protocolo de adaptación y control de enlace lógico).

LMP: link manager protocol (Protocolo del administrador de enlace).

Localización de dispositivo: Para poder establecer un enlace, un dispositivo Bluetooth necesita localizar a los otros dispositivos Bluetooth que están activos dentro de su radio de alcance.

MAN: Red de Área Metropolitana - una red que se extiende a varios edificios en un área que puede ser tan grande como un pueblo o incluso una ciudad.

Maestro: El dispositivo que inicia una conexión y mientras permanece establecida controla todo el tráfico de una pico - red.

Modo aparcamiento: Uno de los modos de ahorro de energía o standby. En modo de aparcamiento, un dispositivo esclavo no participa en la pico - red pero permanece sincronizado con ella. El modo de aparcamiento se utiliza para que pueda aumentar el número de dispositivos esclavos conectados a un maestro.

Nombre de dispositivo: El nombre con el que un dispositivo Bluetooth se identifica a sí mismo ante otros dispositivos.

Notebook: PC portátil que suele tener todas sus funciones, al incluir una pantalla, utilizar un sistema operativo estándar completo como Windows y poder ejecutar aplicaciones estándar, pero que no suele admitir tarjetas adaptadoras.

Paging: servicio para transferencia de señalización o información en un sentido, mediante paquetes, tonos, etc…

Palm: Asistente Digital Personal (PDA) del tamaño de la palma de la mano, con sistema operativo PalmOS. Inventado por Palm Computing y adquirido posteriormente por 3Com.

PAN: Red de Área Personal - por ejemplo el tipo de red establecida por Bluetooth, que asegura la comunicación entre todos los dispositivos electrónicos que rodean a una persona.

PC Card: Describe los tipos de tarjetas que pueden conectarse en las ranuras (slots) de algunos ordenadores, típicamente notebooks, para añadir funcionalidad adicional, como conexión a red, módem o Bluetooth.

PDA: Asistente Digital Personal - un pequeño organizador electrónico, normalmente del tamaño de la palma de la mano, que suele incluir funciones como calendario, agenda, lista de tareas, base de datos de contactos y libreta de anotaciones.

Perfil: Aplicación facilitada por un dispositivo Bluetooth. Para que dos dispositivos se comuniquen entre sí deben compartir el mismo perfil. Por ejemplo, para transferir ficheros desde un ordenador a otro, ambos ordenadores deben tener el mismo perfil de transferencia de ficheros.

Picored: Red ad-hoc del tipo de las que pueden crear dispositivos Bluetooth cuando se encuentran, con el fin de comunicarse entre sí. Red inalámbrica formada por dos o más dispositivos Bluetooth.

PIN: Número de Identificación Personal - como el usado por las tarjetas bancarias.

PPP: Protocolo Punto a Punto - un procedimiento para establecer un enlace punto a punto usando el Protocolo Internet (IP). Es el que suele encontrarse cuando se accede a un Proveedor de Servicio Internet (ISP), pero también puede ser utilizado por Bluetooth para establecer un enlace IP.

Protocolo: Tipo de lenguaje que tienen en común dos o más dispositivos, y que les permite comunicarse entre sí. El ejemplo más conocido es el Protocolo Internet (IP) que deben usar todos los dispositivos conectados a Internet para poder intercambiar información.

Puerto paralelo: Envía datos por varios hilos en paralelo, típicamente a una impresora.

Puerto serie: Envía datos en serie por un solo hilo, típicamente a un módem.

Punto de Acceso: Cualquier punto desde donde se tiene acceso a una red, como un transceptor inalámbrico conectado a una red fija.

RDSI: Red Digital de Servicios Integrados, un sistema inventado en los años 70 que transfiere datos digitalmente por una línea telefónica estándar, al contrario que la señal de voz analógica. La

más común es la RDSI Básica, que ofrece dos canales a 64 Kbit/seg y un canal de control a 16 Kbit/seg.

RTPC (PSTN): Red Telefónica Pública Conmutada.

Salto de frecuencia: Describe un sistema de comunicación basado en ondas radio cuyas frecuencias cambian (saltan) a lo largo del tiempo. Por ejemplo, las frecuencias usadas en Bluetooth saltan 1600 veces por segundo. Su ventaja es que es menos susceptible a interferencias y proporciona mayor seguridad a la señal.

Scatternet: Un grupo de pico - redes conectadas entre sí cuyas áreas de cobertura se solapan.

SCO: synchronous connection oriented (Enlace síncrono orientado a la conexión).

SDAP: service discovery aplication protocol (perfil de aplicación de descubrimiento de servicio).

SDP: service discovery protocol (protocolo de descubrimiento de servicio).

Secuencia directa: Un método de transmisión inalámbrica que puede proporcionar velocidades de transmisión más elevadas y mayor robustez que la técnica de salto de frecuencia en entornos con muchas interferencias radio. El inconveniente es su coste más elevado y que puede soportar a menos usuarios en un área determinada.

SIG: special interest group (Grupo de Interés Especial en Bluetooth).

Síncrono: Flujo de datos que se transmite asociado y a la velocidad de una señal de reloj - por ejemplo en las líneas dedicadas.

SOCKET: es una abstracción de red para los terminales de un canal. El Socket está asociado con el protocolo; usualmente, el PF_INET es usado para asociar un socket con el protocolo TCP/IP.

Symbian: Una aventura conjunta entre Psion, Ericsson, Nokia y Motorola para promocionar el sistema operativo EPOC en dispositivos informáticos inalámbricos.

TCP: Protocolo de Control de Transporte - se refiere a la capa por debajo de la IP que permite a los dispositivos conectados a Internet comunicarse y pasarse mensajes.

TDMA: Acceso Múltiple por División en el Tiempo. Estándar usado en las redes inalámbricas.

Tecnología abierta: Término flexible que se usa a menudo para describir una tecnología informática - normalmente software – cuyo código fuente está abierto para que cualquiera lo pueda modificar como más le convenga, o que ha sido desarrollado por un comité formado por miembros de una asociación industrial informática.

Teléfono celular: Teléfono móvil que funciona usando frecuencias de radio dispuestas en células, como el popular teléfono GSM.

Teléfono inalámbrico: Teléfono que se conecta sin cables, generalmente en el hogar.

Teléfono inteligente: Terminal GSM con pantalla mejorada y nuevas funcionalidades que permiten a sus usuarios acceder fácil y cómodamente a correo electrónico, faxes e intranets empresariales. Los teléfonos inteligentes tienen mayores pantallas, a menudo teclados QWERTY o

pantalla táctil y un software integrado especial para usarse con servicios y aplicaciones específicas, combinando las funciones de teléfono y PDA.

UMTS: Sistema Universal de Telecomunicaciones Móviles - estándar de telefonía celular de tercera generación que usa la banda de 2 GHz y el sistema de paquetes con el objetivo de ofrecer servicios de voz, datos y multimedia con itinerancia global. UMTS promete velocidades de datos de hasta 2 Mbit / seg. Aunque muchos usuarios dispondrán de 144 ó 384 Mbit / seg.

USB: Bus Serie Universal - el intento de la industria informática de estandarizar el hardware PC alrededor de un único y compacto puerto que puede llegar a sustituir a los puertos serie, paralelo y otros diversos tipos de puertos que aparecen en la parte posterior de los PCs.

WAN: Red de Área Extensa - una red que puede llegar a cubrir todo el planeta. Internet podría ser considerada una forma de WAN, aunque más exactamente consiste en múltiples redes entrelazadas - de ahí el término Internet.

WAP: Protocolo de Acceso Inalámbrico - inventado por Nokia, es un sistema estándar que permite a pequeños dispositivos como teléfonos celulares o PDAs presentar datos de Internet a sus usuarios de forma comprensible, lo que se logra principalmente eliminando los gráficos y cambiando el formato del texto. Optimizado para su uso con los canales radio de banda estrecha usados en GSM y GPRS, WAP combinado con Bluetooth es una tecnología clave que puede permitir navegar por Internet desde cualquier lugar.

xDSL: Bucle Digital de Abonado - una tecnología diseñada para proporcionar acceso a alta velocidad a Internet a través de las líneas telefónicas existentes sin que haga falta realizar una llamada ni interrumpir las llamadas en curso. Siempre activo, su forma más común es la de DSL Asimétrico (ADSL), que proporciona un enlace de alta velocidad (hasta 2 Mbit / seg.) hacia el ordenador del usuario, aunque más lento en sentido contrario.

BIBLIOGRAFIA

1. Especificación del Núcleo del Sistema Bluetooth, v1.1, http://www.bluetooth.com
2. Especificación de los Perfiles del Sistema Bluetooth, v1.1, http://www.bluetooth.com
3. Especificación de la API JSR-82 para Bluetooth, http://jcp.org/aboutJava/communityprocess/final/jsr082/index.html
4. Centro de recursos de varias Tecnologías Inalámbricas, http://www.palowireless.com
5. Grupo de Interés Especial para Bluetooth - SIG, http://www.bluetooth.com
6. Arquitectura de Seguridad para Bluetooth, http://www.bluetooth.com/developer/whitepaper/whitepaper.asp
7. Empresa dedicada al desarrollo de aplicaciones software para Bluetooth http://www.rococo.com
8. Compañía dedicada a la distribución de productos para Bluetooth, Incluye la especificación de una API, http://www.digianswer.com
9. Compañía dedicada a la distribución de productos para Bluetooth, http://www.sonera.fi/
10. Pagina de INTEL enfocada en tecnologías inalámbricas, http://www.intel.com/design/mobile/
11. Pagina de NOKIA para la especificación Bluetooth, http://www.nokia.com/nokia/0,5184,397,00.html
12. Pagina de MOTOROLA para la especificación Bluetooth, http://promo.motorola.com/bluetooth/index.html
13. Pagina de ERICSSON para la especificación Bluetooth, http://www.ericsson.com/bluetooth
14. Portal en español dedicado a Bluetooth, http://www.zonabluetooth.com