Descobriment de Serveis en Xarxes Ad-Hoc: Jade Bluetooth Discovery Middleware

Memòria del Projecte Fi de Carrera corresponent als estudis d'Enginyeria Superior en Informàtica realitzat per

Juan Ignacio Toledo Testa i dirigit per Ramon Martí Escalé

Bellaterra, Setembre de 2007

El firmant, Ramon Martí Escalé, professor del Departament d'Enginyeria de la Informació i de les Comunicacions de la Universitat Autònoma de Barcelona

CERTIFICA:

Que la present memòria ha sigut realitzada sota la seva direcció per Juan Ignacio Toledo Testa

Bellaterra, Setembre de 2007

.....
Firmat: Ramon Martí Escalé

Agradecimientos

Agradezco a todo el grupo SeNDA en general y en particular Ramón Martí el haberme brindado la posibilidad de realizar este proyecto en un campo tan apasionante.
Agradezco a mis padres el haberme transmitido curiosidad y ganas de aprender.
Por último agradezco a Silvia por el apoyo y los ánimos que me ha dado durante este proyecto.

Índice

Capítulo 1

Introducción

1.1 Motivación

En el mundo moderno vivimos rodeados, sin ser realmente conscientes de ello, de infinidad de dispositivos dotados de microprocesadores. Desde la caja registradora de los supermercados, hasta los ordenadores pasando por los GPS en el coche, los teléfonos móviles y la televisión digital, no cabe la menor duda de que la tecnología ha llegado a fundirse con nuestra realidad cotidiana.

Estamos en la era de la computación pervasiva y ubicua. En esta situación, el nuevo gran reto tecnológico será conseguir que todos estos dispositivos aunen esfuerzos y colaboren entre ellos para darnos una serie de nuevos servicios antes inimaginables. Se trata de un nuevo ejemplo en el que el total es más que la suma de sus partes.

Para que el trabajo colaborativo de varios dispositivos sea una realidad hay que desarrollar dos pilares básicos. Éstos son la redes Ad-hoc y el descubrimiento automático de servicios. Una red Ad-hoc presenta, en gran parte debido a su topología altamente variable, una serie de nuevos desafíos en comparación con las redes tradicionales.

En las redes Ad-hoc, tiene más sentido que nunca una arquitectura *Peer to Peer* por encima de la tradicional arquitectura Cliente-Servidor. Ésto a su vez, hace cobrar una mayor importancia a los mecanismos automáticos de descubrimiento de servicios.

1.2 Objetivos

El principal objetivo de este proyecto es aportar un pequeño grano de arena a esta próxima revolución y la forma de hacerlo es dotar a la plataforma de agentes móviles JADE (Java Agent DEvelopement) con la capacidad de descubrir servicios en redes Bluetooth. Por tanto, de modo esquemático, los objetivos principales del proyecto serán los siguientes:

- Estudiar y comprender el funcionamiento básico de las redes Bluetooth.
- Estudiar y comprender el funcionamiento del descubrimiento de servicios en redes Ad-hoc según las especificaciones FIPA (Foundation for Intelligent Physical Agents).
- Diseñar y desarrollar un nuevo Discovery Middleware para JADE que permita el descubrimiento de servicios remotos en redes Bluetooth, manteniendo, si es posible, la interfase de comunicación existente entre el DF (Directory Facilitator) y el DM (Discovery Middleware) JXTA.

1.3 Contenido de la memoria

Capitulo 2: Conceptos Básicos

En este capítulo esbozaremos conceptos básicos para la comprensión del proyecto. Se dan cuatro pinceladas sobre redes Ad-hoc y Peer to Peer así como sobre el concepto de agente. También se describen, sin entrar en profundidad, conceptos como el funcionamiento básico de la pila de protocolos Bluetooth o las especificaciones FIPA que más afectan a éste proyecto así como la plataforma JADE.

Capítulo 3: Análisis

En el tercer capítulo veremos los requisitos funcionales y no funcionales del proyecto. Estudiaremos el modelo de mensajes de los DM así como la interfase entre éstos y el DF en el caso particular de JADE. Por último se analizarán las

alternativas existentes en lo relativo a librerías Bluetooth.

Capítulo 4: Diseño

Después de haber visto todos los detalles del análisis y teniendo bien definidas las funcionalidades veremos el diseño de nuestro DM para Bluetooth. Analizaremos los casos de uso más importantes y las clases necesarias para satisfacer los requisitos descubiertos en la fase de análisis.

Capítulo 5: Implementación

Dedicaremos el quinto capítulo de esta memoria a comentar aspectos relacionados con la implementación del proyecto. Básicamente, veremos los detalles de implementación de las clases propuestas en el capítulo anterior.

Capítulo 6: Conclusiones

En el último capítulo haremos una valoración final del proyecto. Revisaremos los objetivos iniciales y propondremos mejoras al Discovery Middleware de Bluetooth y lineas de continuación relacionadas con este proyecto.

Capítulo 2

Conceptos Básicos

2.1 Redes Ad-hoc, Peer to Peer y Agentes Móviles

Una red Ad-hoc es una red creada sin infraestructura previa, donde los nodos se autoorganizan en una topología altamente variable. Podemos tener una red Ad-hoc tanto desde un punto de vista físico como lógico. Un ejemplo típico de redes Ad-hoc físicas serían las MANET (Mobile Ad-hoc NETwork). Estas redes están formadas por dispositivos móviles que se comunican mediante enlaces inalámbricos. Estas redes se caracterizan por una estructura altamente variable, dado que estos dispositivos pueden ir cambiando de una red a otra con gran facilidad. También podemos considerar las redes Ad-hoc desde el punto de vista lógico. Aquí entrarían redes como JXTA o Chord, esta vez la pertenencia a la red Ad-hoc se determina en niveles mas altos de abstracción, donde lo que tradicionalmente sería la capa de aplicaciones nos define una nueva capa de red Ad-hoc.

El término Peer to Peer se refiere a una arquitectura de aplicaciones donde no hay nodos clientes y nodos servidores, sino simplemente nodos, que pueden actuar a la vez como cliente o servidor según las circunstancias. En la actualidad este tipo de aplicaciones han cobrado una gran importancia, todos conocemos aplicaciones de intercambio de archivos basadas en arquitectura Peer to Peer.

Los conceptos de redes Ad-hoc y de arquitectura Peer to Peer se complementan a la perfección. Al tener todos los nodos la capacidad de actuar como cliente o servidor, para acceder a un servicio, tan solo es necesario descubrir otros nodos

sin necesidad de información previa.

Por último, tenemos el concepto de agente inteligente en general y de agente móvil en particular. Abundan distintas definiciones de agente, pero probablemente la que más se ajusta a la realidad es la siguiente: Un agente es una entidad que tiene un objetivo y que interactúa con su entorno para conseguir este objetivo. En nuestro caso, el entorno serían las plataformas de agente. Por último, un agente móvil no es más que un agente con la capacidad de migrar a otro entorno (en nuestro caso, plataforma).

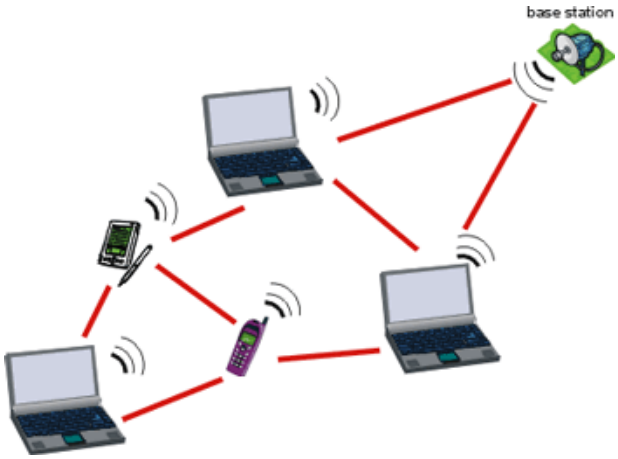

Figura 2.1: Ejemplo de red Ad-hoc.

2.2 Bluetooth

Bluetooth es una pila de protocolos para la comunicación inalámbrica que se diseñó teniendo en mente las ideas de la computación pervasiva y ubicua, por tanto, se puso énfasis en conseguir un bajo consumo y un bajo coste de producción en el hardware necesario. En este escenario cobran gran importancia las PAN (Personal Area Network) así como las redes Ad-hoc inalámbricas.

Teniendo en cuenta esas ideas, se desarrolló la especificación industrial IEEE 802.15.1. Dado que el principal objetivo era tener un bajo coste y consumo, Bluetooth tiene un alcance corto, típicamente de 10 metros (aunque algunos dispositivos pueden alcanzar los 100 metros) y una velocidad de transferencia máxima de 3Mbps. Cabe destacar también que los dispositivos Bluetooth operan en el rango de frecuencias de 2.4Ghz a 2.48Ghz. Este rango de frecuencias es libre, es decir, no es necesaria una licencia para su uso.

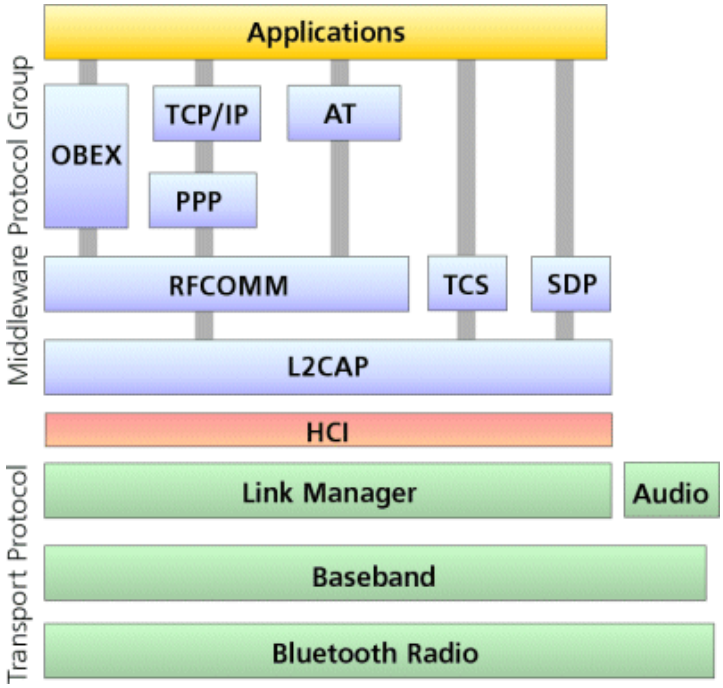

Figura 2.2: Pila de Protocolos Bluetooth.

Sobre este hardware se diseño una pila de protocolos (Bluetooth Stack). En el nivel más bajo tenemos L2CAP (*Logical Link Control and Adaptation Protocol).* Esta capa es la responsable de trabajos de relativamente bajo nivel como pueden ser multiplexar paquetes de capas superiores, controlar la segmentación de paquetes y permite una comunicación unidireccional básica no orientada a conexión.

En la siguiente capa encontramos el primer protocolo de transporte orientado a conexión. Se trata de RFCOMM (Radio Frequency COMMunication). Este

protocolo permite hasta sesenta conexiones simultáneas emulando RS-232. Por este motivo, RFCOMM suele ser conocido como “Emulación de puerto serie”.

El último de los protocolos relevantes para nuestro proyecto es el SDP (Service Discovery Protocol). Con este protocolo se pretendía facilitar el descubrimiento de servicios entre dispositivos Bluetooth. Con ese mismo fin, las empresas responsables de Bluetooth crearon lo que se conoce como perfiles. Un perfil de Bluetooth (Bluetooth *Profile*) no es más que una serie de servicios y protocolos de capa de aplicación.

En el SDP un registro puede contener información diversa sobre el servicio al que representa (nombre, dirección y protocolo de acceso etc..) pero sin duda la más importante es el UUID (Universally Unique IDentifier) un código de 128 bits que identifica dicho servicio. La importancia del UUID radica en que todas las búsquedas de servicio deberán hacerse a partir de este UUID y que por tanto, deberá ser conocido previamente. Esta necesidad de conocer previamente el UUID para acceder al servicio es uno de los motivos de la existencia de los Bluetooth *Profile*s.

2.3 FIPA

FIPA (Foundation for Intelligent Physical Agents) es una organización internacional dedicada a promover la implantación de agentes inteligentes. Su labor principal es la de generar especificaciones que permitan la interoperabilidad entre los agentes.

FIPA, se ha erigido como un estándar en el mundo de los agentes y sus especificaciones cubren gran parte de todo lo que gobierna la “vida” de un agente.

A continuación haremos un breve repaso de las especificaciones más relevantes para entender las bases de este proyecto.

2.3.1 Agent Management Specification

Esta especificación se centra en definir características de las plataformas para asegurar la interoperabilidad.

– AMS (Agent Management System) Es un Agente especial, de implementación obligatoria en toda plataforma FIPA que ofrece el servicio de páginas blancas. Este agente tiene el control de la plataforma y es el encargado de decidir sobre cuestiones como el acceso a la plataforma de agentes externos, permitir la ejecución de ciertos agentes, etc...

– DF (Directory Facilitator) Es otro agente especial, en este caso de implementación opcional. Su función es la de proporcionar el servicio de paginas amarillas. Un agente que ofrezca un servicio puede, voluntariamente, registrar su servicio en el DF de modo que otros agentes puedan encontrarlo. FIPA especifica que el DF, si se decide implementarlo, debe ofrecer como mínimo la posibilidad de registrar, modificar y desregistrar un servicio así como la posibilidad de realizar búsquedas de servicios.

  Opcionalmente, el DF puede ofrecer el servicio de Subscripcion y DesSubscripción que permitirían recibir o dejar de recibir información sobre las altas, bajas o modificaciones del servicio subscrito. Cada registro que tenga el DF con información sobre un servicio recibirá el nombre de DF-Agent-Description. (DFAD)

  Es importante destacar que en esta especificación se define un DF diseñado para entornos estáticos.

– MTS (Message Transport Service) Es el servicio, de implementación obligatoria que permite la comunicación entre agentes de distintas plataformas.

2.3.2 Agent Discovery Service Specification

En esta especificación FIPA, se estudia como dotar al DF de la capacidad de descubrir servicios de otras plataformas a través de redes Ad-hoc. Para poder

aprovechar las características concretas de cada tipo de red, se definen los DM (Discovery Middleware) que estarán bajo el control del DF.

La figura siguiente describe gráficamente el funcionamiento del descubrimiento de servicios. Un agente cualquiera puede consultar al DF sobre otro servicio. Éste hará uso, si es necesario, de los DMs para contactar con DFs de plataformas remotos. Una vez conocido el servicio, el agente puede invocarlo a través de mensajes ACL (Agent Communication Language) vía el MTS.

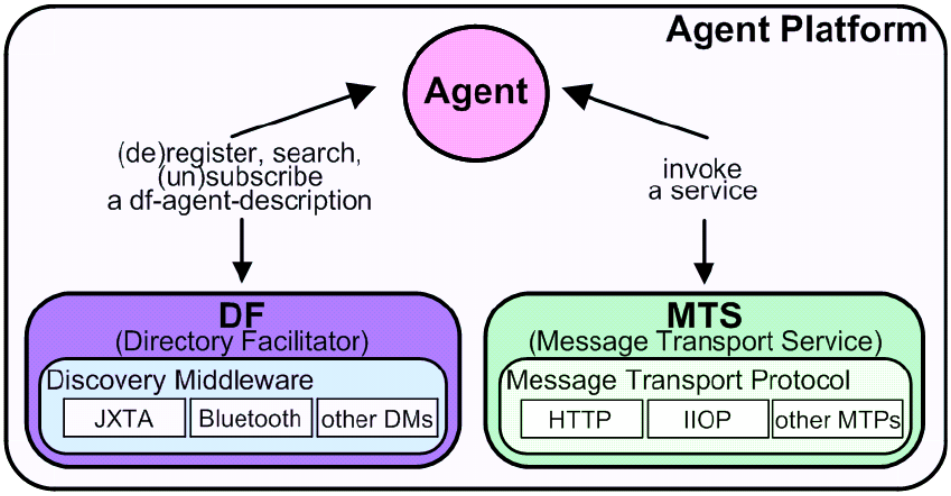

Figura 2.3 Agentes y Servicios en una Plataforma.

Agent Discovery Service Specification define como debe ser la relación entre DM y DF. Son requisitos importantes:

- Un sistema para identificar a los DM
- El DF debe poder poner en marcha y detener a los DMs a voluntad en tiempo de ejecución.
- El DM debe colaborar con el DF para realizar sus funciones en el caso de servicios remotos, que se encuentren en plataformas de la red Ad-hoc.

2.4 JADE

JADE (Java Agent DEvelopement) es un framework para facilitar el desarrollo de aplicaciones basadas en agentes móviles. JADE sigue las especificaciones FIPA y entre sus características más importantes se encuentran el estar codificado íntegramente en JAVA y el ser distribuido bajo licencia LGPL.

Al estar codificado en JAVA, JADE permite abstraer la plataforma de agentes del dispositivo físico. Existe incluso una versión reducida de JADE, conocida como LEAP. Esta versión hace un uso del API de JAVA J2ME (Java 2 Mobile Edition) y por tanto, permite la ejecución de agentes FIPA en dispositivos con baja capacidad de proceso, como podría ser, por ejemplo, un teléfono móvil o PDA.

Capítulo 3

Análisis

Una vez vistos los conceptos básicos necesarios para comprender el proyecto pasaremos a realizar un análisis de los requisitos que debe cumplir nuestro Discovery Middleware de Bluetooth. Partiendo de estos requisitos, analizaremos el funcionamiento tal actual de los Discovery Middleware de JADE así como la interfase existente entre el DF y dichos DM.

3.1 Requisitos funcionales

Según las especificaciones de FIPA, nuestro DM deberá ser capaz de atender los eventos siguientes:

- Registro de un servicio en el DF
- Des-registro de un servicio en el DF
- Modificación de un servicio.
- Subscripción a un servicio.
  - Aviso de registro.
  - Aviso de des-registro.
  - Aviso de modificación.
- Des-subscripción.

Además, FIPA también nos marca los siguientes requisitos:

- Un sistema para identificar a los DM. Esto se traducirá en que el DM debe tener un nombre que el DF debe poder establecer y consultar.
- El DM debe poder ser parado y arrancado en tiempo de ejecución.

3.2 Requisitos no funcionales

El objetivo de este proyecto es realizar un DM para JADE. Por tanto, entre los requisitos no funcionales tendríamos los siguientes:

- Lenguaje de programación JAVA
- Respetar la interfase existente en JADE entre DF y DM.
- El DM debe funcionar en las todas las plataformas en las que funcione JADE.

Al no haber una especificación FIPA sobre el DM Bluetooth no hay restricciones especiales sobre asuntos como la codificación de los mensajes, el tipo de servicio o protocolo concreto a utilizar, por lo tanto, éstos deberán diseñarse desde cero en este proyecto.

3.3 Discovery Middleware en JADE

En proyectos previos como el Discovery Middleware de JXTA [Ore06] y la adaptación del DF a entornos Ad-hoc [Ezp07] se diseñó la actual interfase entre DF y DM. A continuación describiremos su funcionamiento y estudiaremos en que medida dicha interfase se adapta a la red Bluetooth.

Como se ha descrito con anterioridad el DF controlará a cada uno de sus DM y podrá requerir de sus servicios básicamente cuando se produzca un evento. Estos eventos pueden ser un registro, búsqueda, modificación, subscripción y des-subscripción.

3.3.1 Subscripción.

Comenzamos por el servicio de subscripción, puesto que al ser el más complejo condiciona el funcionamiento del resto de servicios.

En el momento en el que un agente realice una subscripción, su DF almacenará dicha subscripción y propagara un aviso de subscripción por toda la red.

Al recibir un DF remoto un aviso de subscripción, buscará entre los servicios registrados localmente. En caso de haber coincidencia, informará a la plataforma original y guardará la subscripción marcándola como subscripción remota. El motivo para guardar esta subscripción es poder avisar posteriormente a la plataforma original si se produce algún cambio en el servicio subscrito.

En el caso de que el DF remoto no encuentre ningún servicio local que coincida con el aviso de subscripción, simplemente lo ignorará.

Cabe destacar también que las subscripciones remotas tendrán un tiempo de validez. Una vez transcurrido este tiempo, el DF enviará al DF original un mensaje de Polling. Si el DF original aun conserva la subscripción local, deberá responder con un mensaje de subscripción.

3.3.2 Registro

Cuando un agente registre un servicio, el DF propagará por la red Ad-hoc un aviso de registro con el DFAD correspondiente. Si este aviso llega a una plataforma que contiene una subscripción local que coincida con el nuevo servicio registrado, ésta informará a la plataforma original para que guarde la subscripción como remota.

En caso de no existir subscripciones locales, el aviso, una vez más, se ignora.

3.3.3 Modificación y Desregistro

En el momento en el que un agente quiera realizar una modificación o

desregistrar un servicio, el DF buscará en su lista de subscripciones remotas. Si existe alguna subscripción remota, éste enviará un aviso al DF remoto responsable de la subscripción. Si por el contrario, no existiera ninguna subscripción remota, no se enviaría ningún mensaje.

3.3.4 Búsqueda

Por último, el caso más sencillo, la búsqueda. Para realizar una búsqueda se propagará un mensaje de búsqueda por la red. Aquellos DF que tengan registrado algún servicio que coincida contestarán a la plataforma original.

3.3.5 Conclusiones:

Al diseñarse este comportamiento del DF y DM se tuvo en cuenta el minimizar el número de mensajes intercambiados. Especialmente intentando evitar el Broadcast. Ésto encaja a la perfección con las características de la red Bluetooth.

Es más, si analizamos a consciencia este diseño, podemos comprobar que escala muy bien. La información sobre los servicios se mantiene siempre local, por lo que un DF apenas incrementará sus necesidades de espacio en memoria por el hecho de pertenecer a una red Ad-hoc de plataformas. En realidad, la única información extra que guarda cada plataforma son las subscripciones remotas que puedan existir para servicios locales. Esta característica también es altamente deseable en las redes basadas en Bluetooth, dado que muchos de los dispositivos Bluetooth son procesadores de bajo coste y, por tanto, de potencia limitada.

Por tanto, se puede concluir que el sistema de funcionamiento DF-DM actual de la plataforma JADE, se ajusta completamente a las redes Bluetooth.

3.4 Librerías Bluetooth

En este apartado analizaremos las opciones disponibles para el uso de librerías Bluetooth. Desde un punto de vista académico es innegable lo interesante del

software libre. A eso hay que añadir las implicaciones morales de trabajar con software libre. Por estas dos razones se optó desde un primer momento por trabajar bajo Linux, teniendo en cuenta que, al estar codificado en JAVA nuestro DM debería ser capaz de funcionar en todas las plataformas en las que funcione JADE.

3.4.1 Bluetooth y Linux

Como hemos visto en el capítulo 2, las especificaciones de Bluetooth cubren desde la capa física hasta la capa de transporte. Este conjunto de protocolos recibe el nombre de Bluetooth Stack. En Linux tenemos, como Bluetooth Stack oficial, BlueZ escrito en C e incorporado en el kernel desde la rama 2.6.

BlueZ incluye una serie de utilidades para el uso de los dispositivos Bluetooth y da soporte a algunos *Profiles* básicos como pueden ser el HID (Human Interface Device), es decir, teclados, ratones etc... BlueZ también nos proporciona unas librerías en C que nos permiten crear aplicaciones capaces de utilizar dispositivos Bluetooth.

3.4.2 Bluetooth y Java

Java 2 Standar Edition (la versión sobre la que está escrito JADE) no incluye soporte nativo para Bluetooth. En J2ME, sin embargo, se incluye JSR 82, una API diseñada conjuntamente por Sun y Motorola. JSR 82 Se ha convertido en el estándar *de facto* para las comunicaciones Bluetooth en Java. Esta implementación incorpora los paquetes *javax.bluetooth* y *javax.obex*.

A pesar de no existir soporte nativo para Bluetooth en J2SE, encontramos varias implementaciones realizadas por terceros que se ajustan (en mayor o menor medida a JSR 82). Podemos ver sus principales características en la siguiente tabla.

Nombre	Licencia	S.O	Java	javax.bluetooth	javax.obex	Comentarios
API's de Java con pila de Bluetooth incorporada						
Harald	Gratuito	Todos	Todas	No	No	Requiere javax.comm.Solo soporta L2CAP
aveLink	Comercial	Todos	Todas	Sí	Sí	
JavaBluetooth	GNU	Todos	Todas	Sí	No	Requiere javax.comm. Solo soporta L2CAP
API's de Java que se ejecutan sobre pilas Bluetooth Externas						
avetana Bluetooth	Comercial	XP, Linux, MacOS	J2SE	Sí	Sí	25€ Las versiones de MacOS y Windows. Versión GPL para Linux
Blue Cove	GNU	XP	J2SE	Sí	No	Soporta L2CAP y RFCOMM
JBlueZ	GNU	Linux	J2SE	No	No	Solo soporta descubrimiento básico
Impronto DK	Comercial	Linux	J2SE	Sí	Sí	Gratis para uso académico.

Figura 3.1: Comparativa de implementaciones JSR 82 en J2SE

Como podemos ver, las implementaciones se dividen básicamente en dos grupos. Las API's que implementan completamente la pila de Bluetooth y las API's que utilizan pilas de Bluetooth externas. Las primeras ofrecen la ventaja de funcionar en cualquier sistema operativo pero tienen el inconveniente de tomar el control del dispositivo Bluetooth, impidiendo el acceso a otras aplicaciones. Las segundas tienen la desventaja de ser dependientes del sistema operativo pero, por ese motivo, no bloquean el acceso al dispositivo Bluetooth para otras aplicaciones.

Sería deseable que la implementación que usemos en este proyecto implementase javax.bluetooth por los motivos expuestos anteriormente. Desearíamos también que fuese una aplicación libre y compatible con varios sistemas operativos. Teniendo en cuenta estos factores, los mejores candidatos son JavaBluetooth y avetanaBluetooth. Nos decantamos por utilizar avetanaBluetooth por varios motivos:

- La realización del proyecto es bajo Linux, por lo tanto podemos usar la versión GPL.
- Al interactuar con la pila Bluetooth del S.O no impide el acceso a Bluetooth de otros programas.
- Esta implementación está basada en JavaBluetooth y JBlueZ. Mejorando la primera que lleva años sin modificarse. Y aportando el indispensable soporte para RFCOMM.
- Soporta javax.obex

El hecho de que las versiones para MacOS y Windows de avetanaBluetooth sean de pago no resulta algo crítico, puesto que, al implementar estas librerías el *estándar* JSR-82 su funcionalidad debería mantenerse para otras implementaciones sin necesidad de modificaciones.

3.4.3 La librería AvetanaBT

Esta implementación del API JSR82, envuelve mediante librerías en java las llamadas al API de la Bluetooth Stack del Sistema Operativo. En el caso de Linux, ésto se consigue con el uso de JNI (Java Native Interface) y las librerías de la pila Bluetooth oficial de Linux BlueZ.

Al estar BlueZ licenciado bajo licencia GPL, podemos disponer de todo el código de avetanaBT para Linux. Estas librerías tienen varias mejoras al margen del las especificaciones JSR 82 sin embargo, no podrán ser utilizadas para ajustarse perfectamente a JSR 82 y mantener así la interoperabilidad.

3.5 Alternativas de representación de los servicios en la red Bluetooth

Gracias al protocolo SDP *(Service Discovery Protocol),* de implementación obligada por cualquier dispositivo Bluetooth, cualquier dispositivo Bluetooth

debería ser capaz de descubrir qué servicios ofrece otro dispositivo.

Cuando una aplicación quiere dar de alta un servicio Bluetooth, debe registrarse en el SDP local y abrir un socket para esperar conexiones. Para identificar cada servicio Bluetooth se utilizan UUIDs (Universally Unique IDentifier) de 128 bits.

Para buscar un servicio, conociendo su UUID, disponemos de dos opciones. En el primer tipo de búsqueda, se buscará el UUID solicitado en cualquier dispositivo Bluetooth cercano y obtendremos solamente la primera coincidencia. En el segundo tipo de búsqueda, buscamos el UUID en un solo dispositivo. Ésto nos permite, en el caso de que haya varios dispositivos en el entorno que ofrezcan el mismo servicio, escoger cuál de ellos deseamos utilizar. Para realizar este tipo de búsquedas, debemos conocer la dirección Bluetooth del dispositivo en el que queremos buscar.

Bluetooth también ofrece la posibilidad de descubrir todos los dispositivos Bluetooth cercanos. De este modo podríamos, posteriormente, realizar una búsqueda de servicios individual en cada dispositivo. Obteniendo todos los servicios del tipo deseado disponibles en la red.

Sin duda, SDP ofrece una primera aproximación al descubrimiento de servicios que es interesante estudiar e intentar aprovechar en este proyecto. A continuación analizaremos las opciones de las que disponemos para representar los servicios de nuestra plataforma en la red Bluetooth.

3.5.1 Un servicio Bluetooth por cada servicio de agente.

Una primera aproximación, bastante intuitiva consistiría en dar de alta un servicio Bluetooth local por cada servicio registrado en el DF local. Las búsquedas de servicios de agentes Jade se podrían traducir a una búsqueda de servicio Bluetooth.

Ésta alternativa resulta atractiva por su sencillez conceptual y porque nos permitiría realizar búsquedas rápidas de un servicio concreto, siempre que solo necesitemos un servicio. Si se desease descubrir todos los servicios de un tipo

disponibles en la red, tendríamos que pasar por el paso previo del descubrimiento de dispositivos y perderíamos esta aparente ventaja.

Sin embargo, el principal inconveniente de esta alternativa consistiría en encontrar una representación válida de los servicios de agente como servicios Bluetooth. La búsqueda en Bluetooth se realiza por UUID, por lo tanto se produce una coincidencia todo-nada. Es decir, si el UUID coincide hemos encontrado el servicio. Por tanto sería extremadamente difícil trasladar una búsqueda semántica como la que propone FIPA para sus servicios a una simple comparación de UUID.

Ciertos expertos demandan que se modifique el mecanismo de descubrimiento de servicios incorporado de Bluetooth para permitir este tipo de búsquedas, pero en la actualidad esto no es posible, por lo tanto esta opción resulta inviable.

3.5.2 Una plataforma es un servicio.

Tras analizar la inviabilidad de la primera opción y comprobar que una búsqueda de servicios debe ir precedida de un descubrimiento de dispositivos proponemos la aproximación de representar la plataforma como un servicio Bluetooth. Los pasos a seguir serían los siguientes:

- Descubrimiento de dispositivos cercanos.
- Comprobar cuales de estos dispositivos ofrecen el servicio de plataforma de agentes.
- Iniciar una comunicación, a nivel de aplicación, con las plataformas remotas que serán las encargadas de decidir cuando hay coincidencia entre el patrón de búsqueda y un servicio de agente.

Esta segunda opción nos da la flexibilidad que necesitamos para implementar nuestra búsqueda semántica y para ofrecer una gestión eficiente de las subscripciones. Además nos permite conocer en que dispositivos Bluetooth hay una plataforma.

Como efecto secundario de establecer esta correlación entre *peer* y plataforma se abre las puertas al uso de servicios propios de la red Bluetooth (que trabajan

en un esquema centrado en los *peers*). Esto queda fuera de los objetivos de este proyecto, pero brinda unas interesantes posibilidades de expansión.

Vistas todas estas ventajas, optamos por esta segunda alternativa, al activar el DM Bluetooth se dará de alta un servicio en el SDP que representará a la plataforma, o si se quiere al DM de esta plataforma. Dicho servicio tendrá una conexión abierta para recibir peticiones e información del sus plataformas vecinas.

Capítulo 4

Diseño

En este capítulo veremos los detalles relativos al diseño del Discovery Middleware para Bluetooth. Estudiaremos los casos de uso de nuestro DM y las clases que necesarias para implementarlo. Propondremos también una estructura sencilla de mensajes entre los DM Bluetooth acorde con la filosofía de FIPA.

4.1 Casos de uso

Tras completar la fase de análisis del proyecto y teniendo claros los requisitos así como la interfase de comunicación que deseamos tener con el Directory Facilitator, podemos pasar a analizar los casos de uso de nuestro Discovery Middleware.

Veremos que en el sistema básicamente tendremos dos actores. El DF que gestionará a nuestro DM y el resto de DMs que pueden enviarnos mensaje. El Directory Facilitator debe ser capaz de iniciar y parar nuestro DM a voluntad. También el DF puede solicitar el envío de un mensaje al resto de plataformas remotas. Estos mensajes podrán ser búsquedas, avisos de registro o mensajes relacionados con la gestión de subscripciones como pueden ser el aviso de modificación de un servicio subscrito, el desregistro de un servicio subscrito, el polling para informar de la caducidad de una subscripción o la cancelación voluntaria de subscripciones.

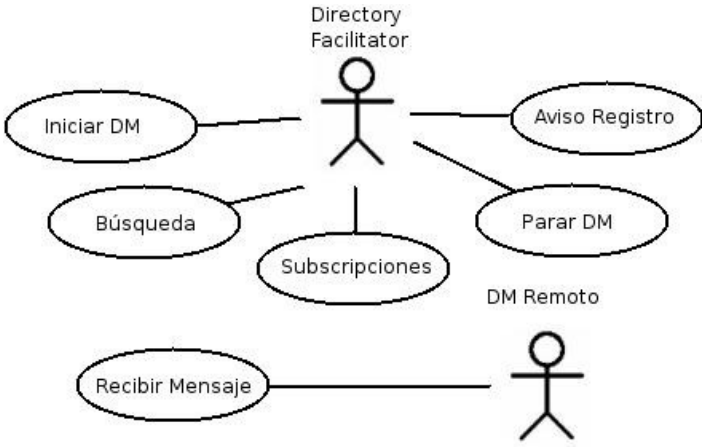

Figura 4.1 Casos de uso del DM Bluetooth

4.1.1 Iniciar DM

El objetivo de este caso de uso es dejar el DM listo para funcionar. Al margen de crear el propio objeto cuyas funciones serán invocadas por el Directory Facilitator, iniciar el DM consistirá principalmente en realizar los pasos necesarios para que el resto de plataformas puedan descubrir que nuestro dispositivo ofrece el servicio Plataforma.

En la siguiente figura podemos ver los tres pasos claves a seguir en el caso de uso Iniciar DM. En primer lugar debemos hacer nuestro dispositivo visible en la red Bluetooth. A continuación debemos dar de alta nuestro servicio en el SDP para poder ser identificados como una plataforma. Por último debemos abrir un canal de comunicaciones para permitir la llegada de mensajes desde las plataformas remotas.

Figura 4.2 Caso de uso Iniciar DM

4.1.2 Búsqueda

El caso de uso búsqueda, es uno de los más complejos. En primer lugar debemos codificar el mensaje, tomando como base los objetos que nos proporciona el Directory Facilitator en su llamada.

Una vez hemos codificado el mensaje buscaremos plataformas a las que enviar nuestro mensaje de búsqueda. Esto se traduce en un descubrimiento de dispositivos Bluetooth cercanos, seguido de un descubrimiento del servicio que representa a las plataformas.

Tras haber confeccionado nuestra lista de plataformas vecinas, deberemos ir contactando con cada una de ellas, realizando la búsqueda. Recibiremos la respuesta de la plataforma y la decodificaremos, acumulando las DF Agent Description que recibamos. Con cada DFAD recibido comprobaremos si hemos alcanzado el número máximo de respuestas especificado a través de las restricciones de búsqueda. En caso de no haber alcanzado dicho valor, contactaremos con la siguiente plataforma y repetiremos el proceso hasta recorrer todas las plataformas.

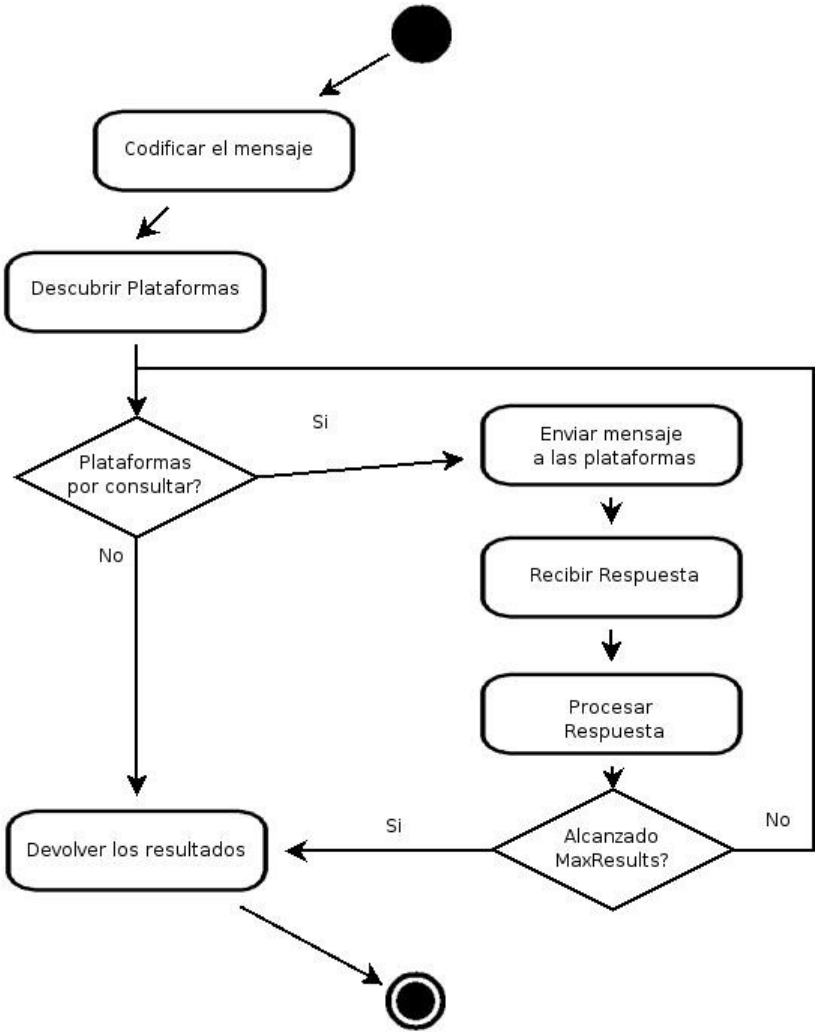

Figura 4.3 Caso de uso Búsqueda.

4.1.3 Aviso de Registro

Para enviar un mensaje de aviso de registro el primer paso será construir el mensaje a partir del DF Agent Description que nos proporcionará el Directory Facilitator. Como hemos visto en la etapa de análisis, para conseguir la funcionalidad de subscripción, estos mensajes de aviso de registro serán propagados a todas las plataformas vecinas.
Para poder propagar el mensaje por todas las plataformas, antes de proceder a su envío deberemos realizar el descubrimiento de plataformas. Una vez tenemos toda la lista de plataformas nos limitaremos a contactar con cada una de ellas y hacerle llegar el aviso.

4.1.4 Subscripciones

Los avisos relacionados con las subscripciones pueden ser de varios tipos. El primer paso para enviar cualquier tipo de mensaje relacionado con las subscripciones será identificar el tipo de mensaje que queremos enviar y codificarlo en consecuencia. Por ejemplo, los avisos de modificación son un caso especial, dado que incluyen dos DF-Agent Description.

En el caso de que tengamos que enviar un mensaje de solicitud de subscripción, lo enviaremos a todas las plataformas del mismo modo en el que hemos visto anteriormente para los avisos de registro, es decir, realizando el descubrimiento previo de plataformas.

El resto de los mensajes de gestión de subscripciones se enviarán únicamente a aquella plataforma a la que se quiera notificar. Para conseguirlo, en los mensajes de aviso de registro y de búsqueda, la plataforma se identifica y nos proporciona la información necesaria (en modo de URL de servicio Bluetooth) para poder contactar con ella posteriormente. A partir de esta URL contactaremos directamente y enviaremos el mensaje a la plataforma interesada.

4.1.5 Recibir Mensaje

Este caso de uso representa el evento que se produce al recibir un mensaje desde un DM remoto. El primer paso al recibir un mensaje sería analizar dicho mensaje y recomponer la información que hay en el. El tipo de información que podemos encontrar es la identificación del DM remitente, un DF Agent Description (o dos en el caso de una modificación), una serie de restricciones de búsqueda, etc... Al “parsear” el mensaje, crearemos los objetos necesarios para poder interactuar con nuestro Directory Facilitator.

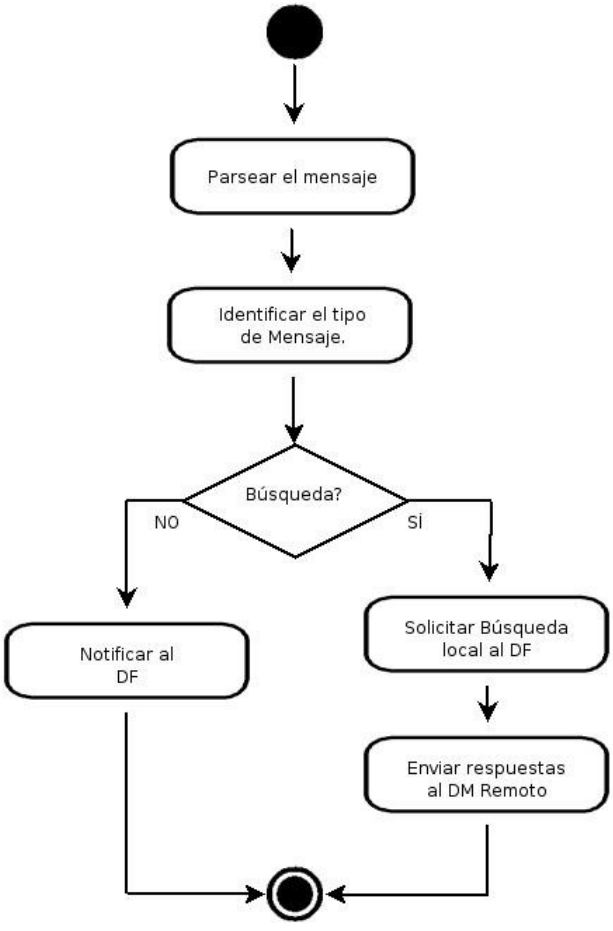

Figura 4.4 Caso de uso Recibir Mensaje

Como hemos visto durante el análisis, la mayoría de mensajes no requieren ningún tipo de respuesta y nos limitaremos a notificar la recepción a nuestro DF. En el caso de que el mensaje sea de búsqueda, sin embargo, realizaremos una búsqueda en nuestro DF y enviaremos los resultados al DM que nos ha realizado la consulta. Podemos ver gráficamente todos los pasos a seguir en el siguiente diagrama.

4.1.6 Parar DM

Este último caso de uso consiste en desconectar nuestra plataforma de la red Bluetooth. Esto consistirá en eliminar el servicio que representa a nuestra plataforma en el SDP de nuestro dispositivo Bluetooth. Los pasos a seguir quedan detallados en la siguiente figura.

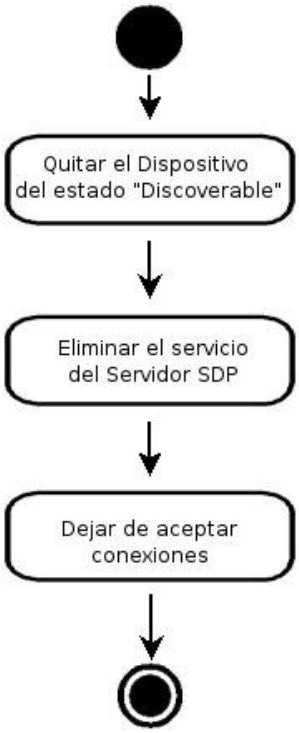

Figura 4.5 Caso de uso Parar DM.

4.2 Clases

A continuación haremos una breve descripción de las clases en las que se estructurará el proyecto. Esta división que proponemos viene dada en parte por la estructura de JADE y por el funcionamiento del API de Bluetooth en JAVA JSR82.

DeviceDiscoveryListener

En JSR82, se define la interfase DiscoveryListener. Al realizar cualquier tipo de descubrimiento, debe pasarse como parámetro un objeto DiscoveryListener. Al producirse un evento, se dispara una función de dicho objeto.

DeviceDiscoveryListener será la encargada de atender básicamente a dos eventos: deviceDiscovered y InquiryCompleted. Su respuesta al primero de ellos consisitrá en acumular los dispositivos descubiertos. Al producirse el segundo evento, inquiryCompleted, la clase permitirá acceder a la lista completa de los dispositivos descubiertos.

ServiceDiscoveryListener

Esta clase tendrá una funcionalidad muy similar a la anterior, con la diferencia de que está dedicada a tratar el descubrimiento de servicios. Atiende al evento serviceDiscovered acumulando los servicios detectados y permitiendo el acceso a ellos una vez se haya producido el evento serviceSearchCompleted.

DMService

La clase DMService es la encargada de la codificación y decodificación de los mensajes entrantes y salientes al DM. Al tener que tratar con varias estructuras, contará con varias variables miembro, que representarán cada una de las partes que puede contener un mensaje que intercambien dos DM. Esta clase no debería interactuar con la red Bluetooth y eso permitiría que se pudiese aprovechar para codificar mensajes en futuras implementaciones de DM si desean usar el formato de mensajes propuesto en este proyecto.

DMVocabulry

Para la realización de este proyecto será necesario ampliar el vocabulario definido por FIPA para dar cabida a mensajes como el Polling o el aviso de modificación de un servicio subscrito. La clase DMVocabulary definirá ciertos Strings estáticos que facilitaran la lectura y el mantenimiento posterior del código.

BluetoothDMServer

Dividiremos los casos de usos vistos anteriormente en dos clases. La primera de ella será la encargada del caso de uso “Recibir Mensaje”. Por tanto, su principal tarea será la de atender los mensajes entrantes y decodificarlos, haciendo uso de la clase DMService.

En esta clase se realizará también, gran parte de las labores de Iniciar DM y Parar DM, puesto que estos casos de uso serán los que permitan o dejen de permitir la entrada de conexiones desde otra plataforma.

Para evitar el comportamiento bloqueante de aceptar conexiones, esta clase deberá implementarse como un Thread independiente.

BluetoothServiceManager

Esta clase deberá implementar, gran parte del código del DM. Será la encargada del descubrimiento de dispositivos y de servicios, o lo que es lo mismo, del descubrimiento de plataformas. También será la encargada del envió de mensajes a la red Bluetooth y de gestionar la sincronización entre los diferentes Threads. Esta clase tendrá el control de BluetoothDMServer pudiendo iniciarlo o detenerlo cuando se le solicite.

BTDM

Esta es la clase que envolverá toda la funcionalidad del DM. Implementando la interfase estándar de los DiscoveryMiddleware de JADE y haciendo uso principalmente de las clases BluetoothServiceManager y DMService será la

encargada de crear los mensajes y enviarlos a la red Bluetooth, haciendo el previo descubrimiento de plataformas en caso de que sea necesario.

En la función de búsqueda, esta clase es la responsable de implementar el control de resultados máximos obtenidos, finalizando la búsqueda cuando se haya alcanzado dicho número.

4.3 Estructura de los mensajes:

Dado que no existe una especificación FIPA para los DM de Bluetooth, ha sido necesario para la realización del proyecto definir la estructura de los mensajes que intercambiaran los DM Bluetooth.

Se ha optado por adaptar los mensajes ACL que usan los agentes FIPA para comunicarse entre ellos. Esta adaptación ha consistido principalmente en una simplificación de dicho lenguaje. Estas simplificaciones se han hecho teniendo en mente el minimizar la información transmitida. También ha sido necesario ampliar el vocabulario para dar cabida a ciertas acciones que no habían sido definidas previamente en el lenguaje básico de FIPA, como por ejemplo el Polling.

A continuación analizaremos detalladamente cada uno de los mensajes que, a petición del DF puede enviar o recibir nuestro Bluetooth DM.

4.3.1 Aviso de Registro:

Este mensaje se enviará a todas las plataformas accesibles cuando un agente registre un servicio con su DF. Como la mayoría de los mensajes consta de información sobre el propio DM (nombre y dirección de acceso), información sobre el evento producido y una DF Agent Description con toda la información sobre el servicio registrado.

Ejemplo:

```
( agent-identifier
        :name btdmaid
        :addresses (sequence btspp://0011223344:1 ))
(register
        (df-agent-description
                :protocols (set ExampleProtocol1)
                :languages (set ExampleLanguage1)
                :ontologies (set ExampleOntology1)
                :services (set (service-description
                                :name ExampleServiceName1
                                :type ExampleType1
                                :ownership ExampleOwnership1
                                :protocols (set ExampleServiceProtocol1)))))
```

4.3.2 Subscripción:

Enviaremos este mensaje cuando un agente desee suscribirse a un servicio de una plataforma remota o cuando después de recibir un mensaje de Polling, deseemos mantener la subscripción. Este mensaje puede ser enviado a una plataforma en concreto o ser enviado en broadcast. Normalmente, cuando se envíe en brodacast el DF Agent Description, será en realidad un Template donde se especificarán ciertas características del servicio sobre el que deseamos ser avisados cuando se produzca un alta.

Ejemplo:

```
( agent-identifier
        :name btdmaid
        :addresses (sequence btspp://0011223344:1 ))
(Subscribe
        (df-agent-description
                :protocols (set ExampleProtocol1)
                :languages (set ExampleLanguage1)
                :ontologies (set ExampleOntology1)
```

```
        :services (set (service-description
                :name ExampleServiceName1
                :type ExampleType1
                :ownership ExampleOwnership1
                :protocols (set ExampleServiceProtocol1)))))
```

4.3.3 Cancelar Subscripción:

El mensaje de cancelar subscripción será siempre en Unicast a la plataforma con la que nos hemos subscrito. Se especificará la acción “UnSubscribe” (una de las aportaciones al nuevo vocabulario de DMs) y se adjuntará el DF Agent Description del servicio al que estábamos subscritos.

Ejemplo:

```
( agent-identifier
        :name btdmaid
        :addresses (sequence btspp://0011223344:1 ))
(UnSubscribe
        (df-agent-description
                :protocols (set ExampleProtocol1)
                :languages (set ExampleLanguage1)
                :ontologies (set ExampleOntology1)
                :services (set (service-description
                        :name ExampleServiceName1
                        :type ExampleType1
                        :ownership ExampleOwnership1
                        :protocols (set ExampleServiceProtocol1)))))
```

4.3.4 Polling:

Se enviará, siempre en Unicast, un mensaje de “Polling” a una plataforma remota cuando su subscripción a un servicio local haya caducado. Si ésta desea renovar la subscripción deberá enviar un nuevo mensaje de Subscripción.

Ejemplo:

```
( agent-identifier
        :name btdmaid
        :addresses (sequence btspp://0011223344:1 ))
(Polling
        (df-agent-description
                :protocols (set ExampleProtocol1)
                :languages (set ExampleLanguage1)
                :ontologies (set ExampleOntology1)
                :services (set (service-description
                        :name ExampleServiceName1
                        :type ExampleType1
                        :ownership ExampleOwnership1
                        :protocols (set ExampleServiceProtocol1)))))
```

4.3.5 Notificación de desregistro:

En el caso de que un servicio local, para el que exista una subscripción remota, se desregistre, enviaremos este mensaje a la plataforma responsable de la subscripción.

Ejemplo:

```
( agent-identifier
        :name btdmaid
        :addresses (sequence btspp://0011223344:1 ))
(SubDes
        (df-agent-description
                :protocols (set ExampleProtocol1)
                :languages (set ExampleLanguage1)
                :ontologies (set ExampleOntology1)
                :services (set (service-description
                        :name ExampleServiceName1
                        :type ExampleType1
                        :ownership ExampleOwnership1
```

```
                    :protocols (set ExampleServiceProtocol1)))))
```

4.3.6 Notificación de Modificación:

Este mensaje se enviará a una plataforma remota, que esté subscrita a un servicio local cuando se produzca alguna modificación en el servicio subscrito. En este mensaje se enviará la DF Agent Description nueva y la anterior a la modificación, en ese orden.

Ejemplo:

```
( agent-identifier
        :name btdmaid
        :addresses (sequence btspp://0011223344:1 ))

(SubMod
        (df-agent-description
                :protocols (set ExampleProtocol1)
                :languages (set ExampleLanguage1)
                :ontologies (set ExampleOntology1)
                :services (set (service-description
                        :name ExampleServiceName1
                        :type ExampleType1
                        :ownership ExampleOwnership1
                        :protocols (set ExampleServiceProtocol1)))))
        (df-agent-description
                :protocols (set ExampleProtocol2)
                :languages (set ExampleLanguage2)
                :ontologies (set ExampleOntology2)
                :services (set (service-description
                                :name ExampleServiceName2
                                :type ExampleType2
                                :ownership ExampleOwnership2
                                :protocols (set ExampleServiceProtocol2))))))
```

4.3.7 Búsqueda:

La búsqueda viene representada por un mensaje enviado a todas las plataformas de la red. Dicho mensaje contendrá lo que se conoce como un DF Agent Description Template. Este Template tiene la misma estructura que un DF Agent Description normal, con la peculiaridad de que solo constarán aquellos atributos deseados en el servicio. En este ejemplo vemos como solamente se especifica el tipo de servicio y el protocolo a utilizar.

El mensaje de búsqueda contiene también una serie de restricciones a la búsqueda. Mediante esta serie de restricciones se pueden controlar aspectos de la búsqueda como el número máximo de resultados, la profundidad máxima de la búsqueda o el tiempo máximo de duración de la búsqueda.

Ejemplo:

```
( agent-identifier
        :name btdmaid
        :addresses (sequence btspp://0011223344:1 ))
(search
        (df-agent-description
                :ontologies (set ExampleOntology1)
                :services (set
                        (service-description
                                :type ExampleType1
                                :protocols (set ExampleServiceProtocol1))))
        (search-constraints
                :max-results -1))
```

4.3.8 Respuesta a búsqueda:

Una respuesta a un mensaje de búsqueda será un conjunto de DF Agent Descriptions que coincidan con aquellos campos especificados en el Template original de la búsqueda.

Ejemplo:

```
(set
        (df-agent-description
                :protocols (set ExampleProtocol1)
                :languages (set ExampleLanguage1)
                :ontologies (set ExampleOntology1)
                :services (set
                        (service-description
                        :name ExampleServiceName1
                        :type ExampleType1
                        :ownership ExampleOwnership1
                        :protocols (set ExampleServiceProtocol1))))
        (df-agent-description
                :protocols (set ExampleProtocol2)
                :languages (set ExampleLanguage2)
                :ontologies (set ExampleOntology2)
                :services (set
                        (service-description
                                :name ExampleServiceName2
                                :type ExampleType2 ExampleType1
                                :ownership ExampleOwnership2
                                :protocols (set ExampleServiceProtocol2)))))
```

Capítulo 5

Implementación

A lo largo de este capítulo veremos los detalles de implementación de las clases que hemos propuesto en el capítulo anterior. También describiremos el funcionamiento de los métodos públicos más importantes de dichas clases.

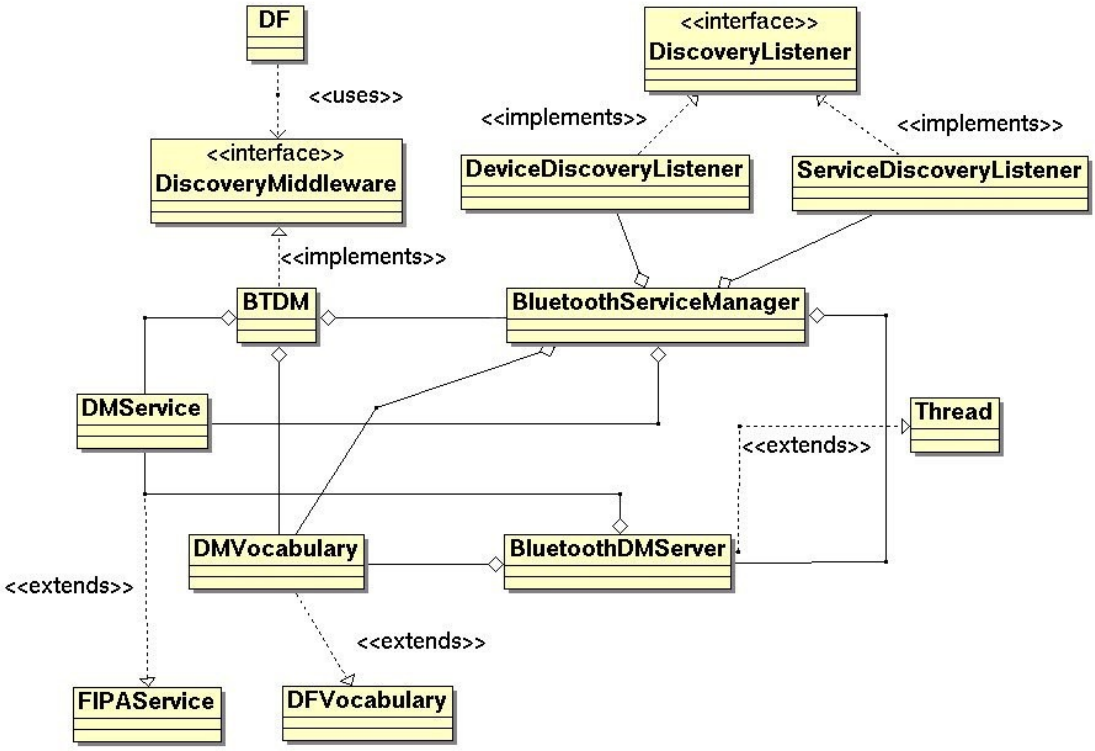

Figura 5.1 Diagrama de Clases

En la figura anterior podemos ver gráficamente la relación que existe entre las clases creadas durante la realización de este proyecto así como algunas de las relaciones con clases. Podemos ver como la clase DF, en este caso una clase de JADE hace uso del interfase DiscoveryMiddleware, diseñado en proyectos anteriores. Nosotros implementamos esta interfase con nuestra clase BTDM.

Podemos ver como nuestra clase BluetoothDMServer extiende a la clase Thread de JAVA, debido a motivos que se explicarán más adelante. Las clases DMService y DMVocabulary extienden a clases de JADE que implementan especificaciones FIPA. Por último, las clases DeviceDiscoveryListener y ServiceDiscoveryListener implementan la interfase DiscoveryListener del API JSR82.

El diagrama de clases anterior solo muestra las relaciones que consideramos más importantes para la comprensión de las clases creadas. Hay muchas otras relaciones entre las clases, que hemos decidido no incluir en este diagrama dado que lo complicarían innecesariamente. Estas relaciones se pueden consultar en la documentación correspondiente, principalmente el API de JSR82 y de JADE.

Una vez tenemos un esquema conceptual sencillo de las relaciones entre las clases del proyecto, pasaremos a describir cada una de ellas y analizaremos sus métodos más relevantes.

5.1 DeviceDiscoveryListener

La clase DeviceDiscoveryListener será la encargada de gestionar el descubrimiento de dispositivos de la red Bluetooth. Esta clase va generando una lista de dispositivos cercanos. Veamos sus métodos para comprender su funcionamiento:

deviceDiscovered

Este método es invocado desde el Bluetooth stack por cada dispositivo que se descubre durante un Inquiry (búsqueda de dispositivos). Su función consiste en agregar el dispositivo a la lista de dispositivos interna de la clase.

inquiryCompleted

El método inquiryCompleted es invocado cuando se ha concluido la búsqueda de dispositivos. Este método desbloquea el acceso a la lista de dispositivos.

getDevices

Una vez se ha completado el inquiry, este método nos permitirá acceder a la lista de dispositivos descubiertos.

5.2 ServiceDiscoveryListener

La clase ServiceDiscoveryListener es la encargada de gestionar el descubrimiento de servicios Bluetooth y es lo que determinará si un dispositivo ofrece el servicio de plataforma de agentes. Su funcionamiento es muy similar a la clase anterior DeviceDiscoveryListener y sus métodos son los siguientes:

servicesDiscovered

Será invocado cuando se encuentre el servicio solicitado. Como en el caso anterior, iremos acumulando los servicios encontrados en una lista.

serviceSearchCompleted

Este método es invocado al finalizar la búsqueda de servicios y desbloquea el acceso a la lista de servicios Bluetooth encontrados.

getServices

Nos retornará la lista completa de servicios descubiertos, una vez haya finalizado la búsqueda de servicios.

5.3 DMService

Como hemos comentado en el capítulo anterior, con esta clase construiremos y decodificaremos los mensajes siguiendo la estructura propuesta en este proyecto. Esta clase tiene como variables miembro, todo lo necesario para representar cualquier tipo de mensaje que puedan intercambiar los DM.

Tendremos dos objetos de tipo DFAgentDescription (necesario el segundo para los mensajes de modificación), uno de tipo SearchConstraints para las búsquedas. Un string que especificará el tipo de mensaje, de entre los definidos

en la clase DMVocabulary y un AID que contendrá el nombre y la dirección de nuestro DM para poder ser contactado posteriormente sin necesidad de descubrimiento previo.

Entre sus métodos públicos encontramos, gets y sets para cada una de sus variables miembro así como los siguientes:

getMessage

Este método tomará los objetos miembro de esta instancia de la clase DMService, que deben haber sido inicializados previamente, y generará el mensaje correspondiente para enviar a la red Bluetooth.

decodeMessage

Este método ofrece la funcionalidad complementaria al anterior, recibirá como parámetro un mensaje y lo decodificará instanciando cada uno de los objetos miembros que encuentre.

decodeDFDSet

Esta última función, es una función estática, gracias a que solo debe devolver un tipo de objetos. Recibirá un mensaje del tipo “respuesta a búsqueda” y nos retornará un array de DFAgentDescription.

5.4 DMVocabulry

La clase DMVocabulary no es más que una colección de Strings estáticos, y por tanto no contiene métodos. Se usa para simplificar la generación de los mensajes. Esta clase extiende a la clase proporcionada por JADE DFVocabulary. Por ese motivo solo ha sido necesario añadir el vocabulario relativo a la gestión de las subscripciones. Por ejemplo, para construir un mensaje de subscripción, utilizaremos el String DMVocabulary.SUBSCRIBE en lugar de un String que contenga la cadena “Subscribe”. Así se facilita la lectura del código y el mantenimiento y se evitan problemas con mayúsculas y minúsculas.

5.5 BluetoothDMServer

Esta clase es la responsable de recibir los mensajes y decodificarlos. Se trata de una clase que extiende a la clase Thread, por tanto se ejecuta como un thread separado que es el encargado de recibir los mensajes.

run

Dentro del método run encontramos la funcionalidad necesaria para recibir los mensajes, éste método es el encargado de permanecer a la escucha de mensajes nuevos. Debido a que el esperar conexiones tiene un comportamiento bloqueante en JSR82, esta funcionalidad ha sido incluida en un Thread independiente.

Al recibir un mensaje, el método run invoca al metodo processMessage de esta misma clase, cuyas funciones vemos a continuación.

processMessage

El método processMessage procesa el mensaje recibido por nuestro dispositivo Bluetooth haciendo uso de los métodos creados a tal efecto en la clase DMService. Una vez identificado el mensaje realiza las acciones pertinentes, que pueden ser identificar al DF o generar una búsqueda local y remitir los resultados al DM remoto.

5.6 BluetoothServiceManager

Esta clase es la principal responsable del funcionamiento del Discovery Middleware de Bluetooth. Sus funciones son variadas y por tanto pasaremos directamente a analizar cada uno de sus métodos.

start

Al llamar al método start se construirán los objetos necesarios para ofrecer la funcionalidad del DM. Este método crea una instancia de la clase BluetoothDMServer e inicia su ejecución como Thread paralelo.

stop

En caso de que se esté ejecutando una instancia del Thread BluetoothDMServer lo detiene, desregistrando el servicio plataforma del servidor SDP local.

findDevices

El método findDevices será el encargado de gestionar la búsqueda de dispositivos Bluetooth. Iniciará un inquiry y creará una instancia de la clase DeviceDiscoveryListener. Esperará a que el inquiry finalice y nos devolverá la lista completa de dispositivos contenidos en nuestro DeviceDiscoveryListener.

findServices

Se trata de un método similar al anterior, recibe una lista de dispositivos y busca en ellos el servicio de plataforma de agentes. Una vez finalizada la búsqueda nos retornará la lista de servicios descubiertos.

findPlattforms

Este método envuelve a los dos métodos anteriores. En primer lugar realiza un descubrimiento de dispositivos y posteriormente una búsqueda del servicio en cada uno de ellos mediante el método findServices. Finalmente genera una lista de strings con la URL de conexión Bluetooth de las plataformas descubiertas.

send

El método send recibe como parámetros una URL de conexión a plataforma y un mensaje. Establece la conexión con dicha plataforma y envía el mensaje. En el caso de que el mensaje provoque alguna respuesta, nos devolverá esta respuesta en forma de string.

sendBroadcast

Por último el método sendBroadcast facilita el envío de mensajes a todas las plataformas y se utilizará para enviar avisos de registro y mensajes de solicitud de subscripción. Este método realiza un descubrimiento de plataformas (findPlattforms) y envía el mensaje que se le pasa como parámetro a cada una de ellas.

5.7 BTDM

Esta clase contiene todo lo que la plataforma JADE necesita para hacer uso del nuevo Discovery Middleware. En la clase BTDM se implementa la interfase estándar de DiscoveryMiddlware para JADE propuesta en proyectos anteriores.

Analicemos uno a uno los métodos públicos de esta clase:

setName

El método setName permite que el DF asigne un nombre al DM para identificarlo, uno de los requisitos de la especificación de FIPA de Agent Discovery Service Specification.

getName

Bastante claro por su nombre, este método devolverá el nombre del DM.

start

Este método representa el caso de uso iniciar DM y es a su vez, el encargado de la construcción de todos los objetos necesarios para el funcionamiento del DM. Básicamente crearía un objeto DMService y un BluetoothServiceManager que a su vez crearía y controlaría un objeto de tipo BluetoothDMServer.

stop

El método stop, será el encargado de detener el DM, para ello solicitaría a BluetoothServiceManager deje de aceptar conexiones y borre el servicio plataforma del servidor SDP local.

sendRegAnnounce

Invocando a este método con los parámetros correspondientes se propagará a todas las plataformas cercanas un aviso de registro.

sendSearch

Para realizar una búsqueda de servicios en la red Bluetooth utilizaremos éste método. A partir de los parámetros especificados por el DF se codificará el mensaje y se enviará a las plataformas una a una hasta recorrerlas todas o haber alcanzado el número máximo de resultados solicitados. Este método implementa el caso de uso búsqueda.

sendSubscribe

Por último el método sendSubscribe sería el encargado de todo tipo de mensajes de gestión de subscripciones. Si al llamar a este método le proporcionamos la

dirección de una plataforma, enviará el mensaje a dicha plataforma. Si no se especifica ninguna dirección, se comportará de un modo similar a sendRegAnnounce, propagando el aviso de subscripción por toda la red.

Capítulo 6

Conclusiones

Ha llegado el momento de valorar todo el proceso de la realización del proyecto y ver que conclusiones pueden sacarse al respecto.
Gran parte del proyecto se ha centrado en estudiar el funcionamiento de las tecnologías base. Se han dedicado varias horas a comprender el funcionamiento de la plataforma de agentes móviles JADE. Una vez ya centrados en aquellos aspectos que afectan más directamente a este proyecto, se ha ahondado en la comprensión del funcionamiento de Bluetooth y el Descubrimiento de servicios propuesto por FIPA. También hemos estudiado el diseño previo de otros DMs para JADE intentando valorar su idoneidad para el caso concreto de las redes Bluetooth.

Desde el punto de vista tecnológico hemos analizado las diferentes alternativas en lo referente a implementaciones del API JSR82. Una vez adquiridos los conocimientos y la comprensión del problema a tratar se ha propuesto un diseño y posteriormente una implementación que encajase de la mejor forma posible.

Al no existir aún especificaciones FIPA sobre como implementar un DM para Bluetooth, ha sido necesario tomar muchas decisiones y diseñar ciertos aspectos intentando seguir el espíritu de FIPA. Este puede ser el caso de decidir representar la plataforma de agentes como un servicio Bluetooth o la codificación de los mensajes inspirada en el lenguaje ACL.

Sin duda es interesante recalcar el hecho de que proyectos como éste, basados en tecnologías jóvenes, como lo son Bluetooth o JADE, son realmente un reto. También produce una gran satisfacción personal el trabajar siguiendo las convicciones morales y aportando algo al mundo del software libre.

Sin embargo, estos aspectos atractivos y motivadores pueden ser un arma de doble filo. En un primer momento hubo ciertas dificultades para conseguir que el dispositivo Bluetooth funcionase bajo Linux. Más adelante, durante la realización del proyecto se detectaron algunos errores en la librería avetanaBT que fueron corregidos, en parte, con una nueva versión publicada a finales de Junio. Sin embargo, persistían algunos errores que nos obligaron a realiazar un estudio detallado del código de la la librería llegando incluso a tener que modificar el código de la misma.

A pesar de todas estas dificultades, o quizás gracias a ellas, no puedo dejar de valorar positivamente la experiencia de realizar este proyecto. Por último me gustaría comentar que ha sido agradable trabajar en un tema, a nuestro entender, tan interesante y actual.

6.1 Revisión de los objetivos iniciales

Al iniciar este proyecto nos marcamos los siguientes objetivos principales:

- Estudiar y comprender el funcionamiento de las redes Bluetooth.
- Estudiar y comprender el funcionamiento del descubrimiento de servicios en redes Ad-hoc según las especificaciones FIPA.
- Diseñar y desarrollar un nuevo DM para JADE que permita el descubrimiento de servicios remotos en redes Bluetooth, manteniendo, si es posible, la interfase de comunicación existente entre el DF y el DM JXTA.

Una vez concluido el proyecto, analizaremos en que grado se han cumplido cada uno de estos tres objetivos principales.
Hemos hecho un estudio del funcionamiento de las redes Bluetooth así como de

las herramientas disponibles en JAVA para hacer uso de dicha tecnología. Estos conocimientos han sido clave en para el éxito del proyecto. Por tanto se puede afirmar que este objetivo ha sido alcanzado.

En lo relativo al segundo objetivo, hemos estudiado las especificaciones de FIPA en lo relativo al Directory Facilitator en entornos Ad-hoc. La comprensión de los conceptos descritos en dichas especificaciones resulta imprescindible para implementar un Discovery Middleware.

La consecución de los dos objetivos anteriores, no era más que un paso previo para el objetivo central del proyecto. Diseñar e implementar un Discovery Middleware para Bluetooth. Hemos propuesto un diseño que respeta la interfase existente entre el DF y el DM en JADE y lo hemos implementado probando la viabilidad de nuestro diseño y reforzando la validez de la interfase propuesta en proyectos anteriores. Este último objetivo ha sido también alcanzado.

6.2 Líneas de continuación

Este proyecto puede proporcionar varias líneas de continuación, dado que toca varios aspectos. En primer lugar tendríamos la posibilidad de mejorar el Discovery Middleware de Bluetooth. Una de las posibles mejoras sería permitir autentificación y cifrado del tráfico entre las plataformas. Otra opción que podría ser interesante sería dar la posiblidad de abrir varios canales simultáneos para la recepción de mensajes. Tener más de un canal abierto podría resultar interesante en algunos casos concretos donde la plataforma Bluetooth genere un gran tráfico, sin embargo probablemente no compensa en la mayoría de dispositivos Bluetooth ya que, debido a su baja capacidad de proceso, no deberían generar demasiado tráfico.

Desde el punto de vista de los Discovery Middleware sería interesante dotar a la plataforma JADE de soporte para otros tipos de redes Ad-hoc, como pueden ser Chord o Gnutella.

Desde el punto de vista de incorporar servicios Bluetooth a la plataforma JADE, probablemente sería interesante ofrecer un sistema de migración de agentes para redes Bluetooth, probablemente un sistema basado en OBEX sería apropiado, así como un sistema MTS para permitir la comunicación entre agentes de distintas plataformas.

Aun hay mucho camino por recorrer para las redes Bluetooth y para los agentes móviles pero no cabe duda de que ambas tecnologías pueden trabajar muy bien juntas.

Bibliografía

[LPJ01] David Holmes, James Gosling, Ken Arnold. El lenguaje de programación java. Prentice Hall, 2001.

[JABWT07] The Java APIs for Bluetooth Wireless Technology, 2007 <http://developers.sun.com/mobility/midp/articles/>

[JBH07] Java Bluetooth Howto, 2007 <http://www.caside.lancs.ac.uk/java_bt.php>

[BTCS04] Bluetooth 2.0 Core Specification, 2004 <http://www.bluetooth.com/>

[AMS04] FIPA Agent Management Specification, 2004 <http://www.fipa.org/docs/output/f-ou5860t-00160/>

[ADSS04] FIPA Agent Discovery Service Specification, 2003 <http://www.fipa.org/specs/fipa00095/>

[FADS06] FIPA P2PNA WorkGroup. Functional Architecture Specification Draft 0.12, 2006 <http://www.fipa.org/subgroups/P2PNA-WG-docs/P2PNA-Spec-Draft0.12.doc>

[Ore06] Orellana, Oriol. JADE Discovery Middleware en JXTA: Projecte de Final de carrera. Bellaterra: Universitat Autònoma de Barcelona, 2006

[Ezp07] Ezponnda, Sílvia. Descobriment de serveis en Xarxes Ad-hoc: Directory Facilitator. Projecte de final de carrera. Bellaterra. Univesitat Autònoma de Barcelona, 2007.

.....

Firmado: Juan Ignacio Toledo Testa

Bellaterra, Septiembre de 2007

Resumen

En este proyecto se ha diseñado e implementado un nuevo Discovery Middleware para la plataforma de agentes móviles JADE. Este DM colaborará con el Directory Facilitator para ofrecer a los agentes un servicio transparente que les permitirá descubrir servicios en plataformas remotas que sean accesibles a través de redes Ad-hoc basadas en tecnología Bluetooth.

Resum

En aquest projecte s'ha dissenyat i implementat un nou Discovery Middleware per a la plataforma d'agents mòbils JADE. Aquest DM col·laborarà amb el Directory Facilitator per tal d'oferir als agents un servei transparent que els permetrà descobrir serveis en plataformes remotes que siguin accessibles mitjançant xarxes Ad-hoc basades en la tecnologia Bluetooth.

Abstract

In this project, a new Discovery Middleware for JADE agent platform has been designed and implemented. This DM will work together with the Directory Facilitator in order to offer agents a transparent service which will allow them to discover services in remote platforms accessible via Bluetooth powered Ad-hoc networks.