PROYECTO FINAL DE CARRERA

Turist@

Agentes al servicio del turista mediante telefonía móvil

Alexandre Viejo Galicia
Ingeniería en Informática

Director de proyecto:
Dr. Antonio Moreno Ribas

Escola Tècnica Superior d'Enginyeria (ETSE)
Universitat Rovira i Virgili (URV)

ÍNDICE

1. INTRODUCCIÓN 4
1.1. FIPA 5
1.2. Grupo de investigación BANZAI - GruSMA 5
1.3. Aplicaciones previas de agentes a servicios turísticos 6
1.4. Objetivos del proyecto 7
1.5. Estructura del documento 7
1.6. Recursos utilizados 8

2. AGENTES Y SISTEMAS MULTI-AGENTE 9
2.1. Propiedades de los agentes 9
2.2. Sistemas Multi-Agente 10
2.2.1. Ventajas de un Sistema Multi-Agente 10
2.2.2. Gestión de un Sistema Multi-Agente 10
2.2.3. Lenguajes de comunicación entre agentes (ACL) 11
2.2.3.1. Componentes de un mensaje 11
2.2.3.2. Protocolos de comunicación 13

3. JADE 14
3.1. Paquetes de JADE 14
3.2. Componentes principales de JADE 15
3.3. Agentes y Comportamientos (Behaviours) 15
3.3.1. Comportamientos Simples 16
3.3.2. Comportamientos Compuestos 16
3.4. Lenguaje de comunicación 16
3.5. Ontología 17
3.6. Protocolos de comunicación 17
3.6.1. FIPA REQUEST 17
3.6.2. FIPA QUERY 18
3.6.3. FIPA CONTRACT-NET 18
3.6.4. Clases proporcionadas por JADE 18
3.7. Herramientas de JADE 19

4. LEAP SOBRE JADE 23
4.1. Entorno de ejecución JADE-LEAP 23
4.2. Ejecución de JADE-LEAP en estaciones (PC o Servidor) 24
4.3. Ejecución de JADE-LEAP en dispositivos con recursos limitados 25
4.3.1. PDAs con PersonalJava 26
4.3.2. Móviles con MIDP1.0 26
4.4. Ejemplo final del entorno JADE-LEAP 27
4.5. Limitaciones en JADE-LEAP para pjava y midp 28
4.6. JADE-LEAP y la conexión sin cables (wireless) 28
4.6.1. Conexión Bluetooth entre PC y Nokia N-GAGE 29
4.6.2. Conexión GPRS entre PC y Nokia N-GAGE 31

5. J2ME 32
5.1. Introducción 32
5.2. La API de MIDP 32
5.3. El Midlet 33
5.4. Interfaz gráfico 34
5.4.1. Interfaz gráfico de alto nivel 35
5.4.2. Interfaz gráfico de bajo nivel 36
5.5. Persistencia de datos 37
5.6. Conexión a redes 37

6. NOKIA N-GAGE 39
6.1. Características técnicas 39
6.2. Soporte J2ME 40
6.3. Soporte Visual C++ 41

7. SMA TURIST@ 42
7.1. Descripción general del aplicativo 42
7.2. Descripción general del funcionamiento 42
7.3. Justificación en la utilización de un SMA 43
7.4. Ámbito de trabajo 44
7.5. Arquitectura del sistema 44
7.6. Funcionalidades de los diferentes tipos de agente 47
7.6.1. Agente Personal 47
7.6.2. Agente de Actividad 47
7.6.3. Agente Broker 47
7.6.4. Agente Recomendador 48
7.7. Comunicación entre agentes 49
7.7.1. Protocolos 49
7.7.2. Ontología 49
7.7.2.1. Objetos 49
7.7.2.2. Acciones 58

8. EL SISTEMA DE RECOMENDACIÓN 62
8.1. El perfil de usuario 62
8.2. Generación del perfil inicial 62
8.3. Actualización del perfil por aprendizaje 63
8.3.1. Actualización explícita del perfil 63
8.3.2. Actualización implícita del perfil 64
8.4. Cuando se recomienda 65
8.5. Cómo se recomienda 65

9. EL SISTEMA DE LOCALIZACIÓN 67

10. DEMOSTRACIÓN PRÁCTICA 70
10.1. Inicialización del sistema 70
10.2. Creación perfil de usuario 70
10.3. Pantalla y menú principal 71
10.4. Recomendación 71
10.5. Búsqueda de actividades 73
10.6. Localización 75
10.7. Avisos del sistema 75
10.8. Agenda y votaciones 76
10.9. Interfaz gráfica de los agentes de actividad 77

11. VALORACIÓN Y CONCLUSIONES 81

12. FUTUROS PROYECTOS 82

13. AGRADECIMIENTOS 83

14. BIBLIOGRAFÍA 84

1. INTRODUCCIÓN

El mundo en que vivimos nos exige de forma imperativa una mayor facilidad a la hora de realizar cualquier tipo de tarea, sea cual sea el ámbito donde nos movemos. Además, no sólo exige facilidad sino también comodidad en el trato, rapidez en la resolución y, sobretodo, eficiencia. Estos tres factores, que a priori podrían parecer antagónicos, son un desafío para las nuevas tecnologías, que responden aportándonos el concepto de *asistente personal electrónico*. El cometido del asistente, es ahorrar tiempo y dinero para conseguir que las personas tengamos una vida más cómoda y llevadera.

Podemos imaginar cientos de situaciones donde tener lo que llamaríamos un asistente informatizado nos sería de gran utilidad. Centrándonos en el ámbito del ocio, podríamos pensar en las facilidades que obtendríamos al planear un fin de semana, siendo el asistente el encargado de realizar todo el trabajo más pesado: buscar espectáculos, reservar entradas, billetes, etc. Y todo ello dándonos un servicio a nuestra medida, totalmente personalizado, dado que el asistente personal nos conoce, sabe nuestros gustos y lo que estamos dispuestos a gastar. Si, por el contrario, quisiéramos enfocar nuestro asistente a un entorno de trabajo, las ventajas también son evidentes, tanto a la hora de buscar información, como planificar sistemas de alarma, detección... Los campos son múltiples, solo haría falta aplicar el asistente personal a ellos, con el software adecuado.

No obstante, el concepto de agente personal se encuentra con un grave problema, y es el de la movilidad. Habremos visto estaciones y ordenadores portátiles ofrecernos su ayuda en los entornos comentados anteriormente, e incluso volverse indispensables en la mayoría de las situaciones cotidianas. Sin duda son eficientes y naturalmente muy rápidos, pero no presentan facilidad en su uso y por descontado no son cómodos de utilizar, incluso en el caso del portátil más ligero, su volumen y peso nos impide considerarlo el asistente personal que nos tiene que acompañar allí donde vayamos.

Es en este punto donde pretendemos mejorar las condiciones existentes, y darle un verdadero impulso al agente personal. Creando de esta forma un asistente que se ejecuta en dispositivos móviles y que da servicio útil y eficiente en situaciones reales.

Se pueden considerar diversas alternativas para construir una plataforma móvil que permita la ejecución de agentes personales, tales como VisualC++ o VisualBasic. Estas opciones, aunque interesantes, no nos aportan las soluciones que nosotros buscamos, y de hecho, se alejan del concepto de asistente personal. Nuestro asistente tiene que ser capaz de comunicarse y colaborar con muchos otros asistentes y herramientas provistas por empresas y gobiernos. Es en este marco donde podemos encontrar una nueva tecnología emergente, nacida de la mano de la Inteligencia Artificial, que trabaja con los llamados *Sistemas Multi-Agente (SMA),* sistemas que engloban un conjunto de programas independientes y autónomos, también llamados *agentes*, diseñados para trabajar juntos para una finalidad común.

Es en esta definición donde podemos ver reconocido el modelo de asistente que deseamos; por tanto, es por este camino por donde avanzaremos, topándonos de inmediato con la problemática de la movilidad. Aunque pensados para entornos distribuidos, con los agentes ejecutándose en diferentes máquinas, los SMA requieren suficientes recursos como para complicar su funcionamiento en los pequeños dispositivos que son el objeto de nuestro estudio. Es por ello que el objetivo prioritario de nuestro trabajo será conseguir mantener un agente en un dispositivo móvil, en nuestro caso un teléfono móvil.

Como ya hemos indicado, uno de los puntos que más nos llama la atención de los agentes es su comunicación elaborada, que les permite trabajar correctamente juntos para la resolución de una tarea complicada. Para garantizar que cualquier agente, desarrollado por uno de tantos programadores del mundo, se entienda con otro agente totalmente independiente y desarrollado a su vez por otra persona,

debemos trabajar con unos mecanismos estándar, que garanticen la interoperabilidad entre los sistemas. Para conseguir esto se necesita una entidad encargada de definir y certificar estos estándares aplicados, que velarán para que diversos SMAs puedan interactuar entre si.

1.1. FIPA

Para definir estándares (tanto de arquitectura, lenguajes, protocolos…) se creó la FIPA[1], una fundación sin animo de lucro nacida con la obligación de establecer las reglas para el diseño e implementación de un SMA para garantizar la interoperabilidad que comentábamos en el apartado anterior. Esta organización inició su andadura en el 1996 haciendo pública su primera especificación, y desde entonces se ha ido imponiendo hasta conseguir ser el estándar seguido *de facto* por un gran número de grupos de investigación académicos e industriales.

Una vez conseguida su meta principal: obtener un grueso de especificaciones útiles a la hora de crear SMAs interoperables. El siguiente paso es conseguir que los agentes vean garantizada la comunicación con tecnologías que no se basan en ellos. Dentro de esta perspectiva, en marzo del 2005, la FIPA se unió a la *IEEE Computer Society* [FIPAandIEEE05], pasándose a llamar *FIPA Standards Committe*.

El *IEEE FIPA Standards Committe* actúa como órgano independiente, con sus propias políticas, estructuras, procedimientos…pero beneficiándose de las grandes ventajas que obtiene al encontrarse bajo la organización de la IEEE Computer Society.

En nuestro grupo de trabajo, hemos partido de las especificaciones aportadas por la FIPA para desarrollar todo un conjunto de proyectos que desean explotar las ventajas de los agentes, siguiendo el modelo impuesto por la FIPA [FIPA05]. En el caso de nuestro estudio, nos hemos valido de una herramienta de desarrollo de SMAs que cumple las especificaciones FIPA; por tanto, nos extenderemos más adelante en los comentarios sobre estas reglas tanto de diseño como de implementación.

1.2. Grupo de investigación BANZAI - GruSMA

El grupo de investigación en el campo de la Inteligencia Artificial llamado BANZAI [Banzai05] forma parte de la Universidad Rovira i Virgili, situada en Tarragona (España). Es uno de los equipos de investigación enmarcados dentro del departamento de Ingeniería Informática y Matemáticas (DEIM) y esta formado por diversos profesores y estudiantes de Doctorado.

BANZAI tiene como una de sus líneas de investigación el diseño de aplicaciones de IA y, dentro de ésta, encontramos el grupo de trabajo *GruSMA*[2] [GRUSMA05] que se encarga de desarrollar diversos proyectos basados en SMAs.

GruSMA proporciona un entorno de trabajo en el cual los alumnos de Informática de la ETSE[3], interesados en las nuevas tecnologías de la Inteligencia Artificial, pueden compartir conocimientos y experiencias en el campo de los Sistemas Multi-Agente. Dicho grupo trabaja en el desarrollo de sistemas utilizando una herramienta para el desarrollo de SMAs llamada JADE, la cual cumple con las especificaciones de la FIPA.

[1] *Foundation for intelligent Physical Agent. Organización impulsora del estándard a nivel mundial para la creación de SMAs*

[2] *Grupo de Sistemas Multi-Agente de la Universitat Rovira i Virgili.*

[3] *Escola tècnica superior de Enginyeria de la URV.*

En este documento, además de hacer una introducción a cómo se desarrolla un sistema multi-agente, se hará también una introducción a cómo se implementan utilizando esta plataforma tanto en su enfoque para servidores clásicos, como en su vertiente más moderna y prometedora: los entornos móviles.

1.3. Aplicaciones previas de agentes a servicios turísticos

El SMA tratado se basa en tres sistemas desarrollados anteriormente en el GRUSMA que perseguían la meta de servir al turismo resolviendo cada uno de ellos diferentes ámbitos del problema. La primera versión del asistente fue desarrollada por Pilar Zapata [Zapata03] con una primera aproximación a los SMA aplicados a la búsqueda de actividades de interés para el turista. Yo mismo realicé ese mismo año una adaptación de dicha solución para su uso en PDAs, acercándonos un poco más a lo que hemos dicho un asistente personal debe aportar. La última mejora realizada, implementada en el 2005, corrió a cargo de Montserrat Batet [Batet05] incluyendo en el proyecto, el uso intensivo de técnicas avanzadas de inteligencia Artificial para la recomendación de actividades. Para la solución del problema, este SMA utilizaba un conjunto de agentes para gestionar las actividades que la ciudad, en nuestro caso Tarragona, ofrece a los turistas como monumentos, itinerarios, museos…Así como la capacidad de dicho sistema de sugerir al usuario según sus gustos, todo ello soportado por agentes independientes que nos daba un comportamiento ágil y muy interesante dado su carácter distribuido.

Este aplicativo de recomendación de actividades coincidía exactamente con la idea que buscábamos para poder lanzar el asistente personal definitivo, puesto que en este entorno y con dichas funcionalidades sabíamos que nuestra aplicación ofrecería un óptimo servicio.

El proyecto realizado, a grandes rasgos, presenta un cliente que utiliza un móvil (un Nokia N-GAGE en nuestro caso), para solicitar información respecto a las actividades de ocio que ofrece la ciudad al turista. Éste podrá realizar consultas sobre tipos específicos de actividades a realizar como monumentos, museos, eventos deportivos, películas proyectadas…Y de igual forma el asistente conociendo los gustos del cliente estará capacitado para realizar sugerencias al mismo, con actividades que puedan ser de su interés. El asistente cuenta con un sistema de localización del que hablaremos más adelante, que permite al cliente saber en que lugar de la ciudad se encuentra, mostrando el mapa del mismo e informando de actividades de interés en la zona.

El SMA ha sido implementado usando la plataforma JADE 3.1, que cuenta con la versión más evolucionada del addon LEAP [4] y que nos ofrece tremendas facilidades a la hora de integrar agentes en dispositivos tan limitados como PDAs o móviles. LEAP básicamente nos permite trabajar con unas librerías menos exigentes, en cuanto a recursos se refiere, permitiéndonos, por tanto, trabajar en los limitados entornos móviles.

El agente que se ejecuta en el móvil ha sido implementado con J2ME[5]. Los agentes situados en estaciones fijas se han desarrollado bajo J2SE[6]. Java ha sido el lenguaje de elección dado su relación directa con JADE, además de ser un lenguaje idóneo para el desarrollo de aplicaciones portables y distribuidas.

[4] *Lightweight Extensible Agent Platform. Addon para JADE que nos permite tener agentes en dispositivos con fuertes restricciones de recursos como móviles o pdas.*
[5] *Java2 Micro Edition. Java enfocado para aplicaciones enmarcadas en dispositivos móviles.*
[6] *Java2 Simple Edition. Java enfocado para las aplicaciones que trabajan en servidores estándar.*

1.4. Objetivos del proyecto

El objetivo principal es dar un paso más en nuestra búsqueda del asistente personal basado en agentes, obteniendo un producto que de verdad ofrezca las funcionalidades requeridas por un usuario real. Las claves de este proyecto son:

1. *Funcionamiento en un móvil sencillo:* Aplicación asequible para la mayoría de la gente.
2. *Conexión GPRS:* Libertad de movimientos para el usuario por toda la ciudad.
3. *Sistema de localización GSM:* El usuario sabe en todo momento donde se encuentra.
4. *Mapas de la ciudad:* El asistente muestra mapas para facilitar la orientación del usuario.
5. *Sistema inteligente de recomendación:* Recomendaciones adaptadas a los gustos del cliente y actualización automática de las preferencias del usuario.

Lo que pretendíamos conseguir era tener un SMA funcionando en un entorno distribuido, en el cual el cliente intervendría utilizando un teléfono móvil. Dicho cliente sería avisado en tiempo real de la presencia de actividades en el lugar donde se encontrase, permitiendo realizar peticiones personalizadas y mostrar al cliente su localización en la ciudad.

Para ello, primero conseguimos obtener la posición a partir del sistema GSM del móvil. Posteriormente establecimos la conexión vía GPRS entre el agente personal y las estaciones fijas y para acabar, entramos en la implementación del aplicativo realizando un exhaustivo estudio de las posibilidades que el entorno J2ME nos ofrece para poder obtener todas las funcionalidades deseadas para el asistente de turismo. Recordar en ese punto que todas las anteriores versiones del aplicativo utilizaban J2SE y por tanto un conjunto más rico en librerías para la aplicación. Para nuestra versión en J2ME hemos tenido que esforzarnos para sacar el máximo partido de las limitadas librerías que nos ofrece y resolviendo en última instancia con ideas imaginativas las limitaciones de otra manera insalvables.

Como último apunte, se han realizado ciertas mejoras en la estructura y funcionamiento partiendo de la base provista por las anteriores versiones del aplicativo.

1.5. Estructura del documento

Este documento pretende explicar con detalle todo el proceso seguido para conseguir los objetivos propuestos, y la forma en que subsanamos los problemas encontrados durante el tiempo de estudio e implementación. Dado que para entender correctamente el trabajo realizado se necesitan unos conocimientos básicos, dedicaremos los capítulos preliminares a introducir todos aquellos conceptos que creemos indispensables para entender correctamente lo aquí expuesto.

La documentación se estructura de la siguiente forma: en el capítulo 2 se hace una introducción a los conceptos básicos sobre Agentes y los Sistemas Multi-Agente. En el capítulo 3 se explica la herramienta de desarrollo y programación de Sistemas Multi-Agente utilizada (JADE). En el capitulo 4 nos extenderemos en la aplicación en sistemas móviles del addon LEAP para JADE, haciendo énfasis en el proceso de comunicación entre la estación principal y el móvil; se comentarán tanto la opción GPRS [Gprs05] como la opción bluetooth [Bluetooth05] que el aplicativo también soporta. En el capítulo 5 comentaremos el entorno J2ME y las librerías que ofrece al programador, destacando las que han sido de especial utilidad a la hora de implementar el asistente personal. El capítulo 6 estará dedicado al análisis del dispositivo sobre el que trabajamos: la Nokia N-GAGE. A partir de aquí nos centraremos en la implementación del aplicativo. EL capítulo 7 será para describir el SMA creado: describiremos la arquitectura, la funcionalidad del sistema, ontología utilizada…. En el capítulo 8 se explica el sistema de recomendación implementado. El capítulo 9 está dedicado al sistema de localización, donde

mostraremos sus puntos fuertes y sus debilidades. El capítulo 10 está destinado a la prueba final del sistema. Por último, destinaremos los capítulos finales a una valoración del trabajo realizado, conclusiones y futuros proyectos,

1.6. Recursos utilizados

Para finalizar esta introducción, hagamos una breve referencia al material que de una u otra forma ha intervenido en la creación de este proyecto, tanto en su fase de estudio, como en la de implementación:

HARDWARE:

- Móvil Nokia N-GAGE con J2ME (MIDP1.0), GSM y GPRS
- PC con procesador AMD 900 mhz,256m ram con windows XP
- PC con procesador intel P4 2.6ghz,256m ram con windows XP
- Conceptronic Bluetooth Dongle

SOFTWARE:

- J2SE 1.4.2 de Sun Microsystems
- J2ME Wireless Toolkit de Sun Microsystems
- JADE (3.1), LEAP (3.1) de Telecom Italia Lab
- Plugfree 2.0 para windows de Fujitsu-Siemens
- Visual Studio 6.0 de Microsoft
- MySQL 4.1.7
- MySQL Control Center
- Symbian 6.1 SDK
- PC-Suite for Nokia N-GAGE
- Nokia Developer's Suite 2.2 for J2ME
- mRouter

2. AGENTES Y SISTEMAS MULTI-AGENTE

Antes de entrar a definir que son los Sistemas Multi-Agente, hemos de centrarnos en los elementos que lo forman: los AGENTES.

Podemos encontrar diversas definiciones de agente, todas ellas recogidas en el *Desenvolupament de Sistemes MultiAgent en JADE, Report de Recerca* [Isern et al., 2001]. De todos modos, nos quedaremos con la explicación ofrecida por el doctor Wooldridge en [Wooldridge02]:

> ➢ Un agente inteligente es un proceso computacional capaz de realizar tareas de forma autónoma y que se comunica con otros agentes para resolver problemas mediante cooperación, coordinación y negociación. Habitan en un entorno complejo y dinámico con el cual interaccionan en tiempo real para conseguir un conjunto de objetivos.

2.1. Propiedades de los Agentes

Las propiedades indispensables para que un agente sea digno de llevar ese nombre son:

- Autonomía: es la capacidad de operar sin la intervención directa de los humanos, y de tener algún tipo de control sobre las propias acciones y el estado interno.

- Sociabilidad/Cooperación: los agentes han de ser capaces de interactuar con otros agentes a través de algún tipo de lenguaje de comunicación.

- Reactividad: los agentes perciben su entorno y responden en un tiempo razonable a los cambios detectados.

- Pro-actividad o iniciativa: deben ser capaces de mostrar que pueden tomar la iniciativa en ciertos momentos.

Otras propiedades destacables serían:

- Movilidad: posibilidad de moverse a otros entornos a través de una red electrónica.

- Continuidad temporal: los agentes están continuamente ejecutando procesos.

- Veracidad: un agente no comunicará información falsa premeditadamente.

- Benevolencia: es la propiedad que indica que un agente no tendrá objetivos conflictivos, y que cada agente intentará hacer lo que se le pide.

- Racionalidad: el agente ha de actuar para conseguir su objetivo.

- Aprendizaje: mejoran su comportamiento con el tiempo.

- Inteligencia: usan técnicas de IA para resolver los problemas y conseguir sus objetivos.

2.2. Sistemas Multi-Agente

Llamaríamos Sistema Multi-Agente o MAS a aquel en el que un conjunto de agentes cooperan, coordinan y se comunican para conseguir un objetivo común.

2.2.1. Ventajas de un sistema Multi-Agente

Las principales ventajas de la utilización de un sistema Multi-Agente son:

– Modularidad: se reduce la complejidad de la programación al trabajar con unidades más pequeñas, que permiten una programación más estructurada
– Eficiencia: la programación distribuida permite repartir las tareas entre los agentes, consiguiendo paralelismo (agentes trabajando en diferentes máquinas).
– Fiabilidad: el hecho de que un elemento del sistema deje de funcionar no tiene que significar que el resto también lo hagan; además, se puede conseguir más seguridad replicando servicios críticos y así conseguir redundancia.
– Flexibilidad: se pueden añadir y eliminar agentes dinámicamente.

2.2.2. Gestión de un sistema Multi-Agente

La administración de agentes establece el modelo lógico para la creación, registro, comunicación, movilidad y destrucción de agentes. En este proyecto vamos a seguir los estándares establecidos por la FIPA. En la figura 1 se puede ver el modelo de administración de agentes para este entorno:

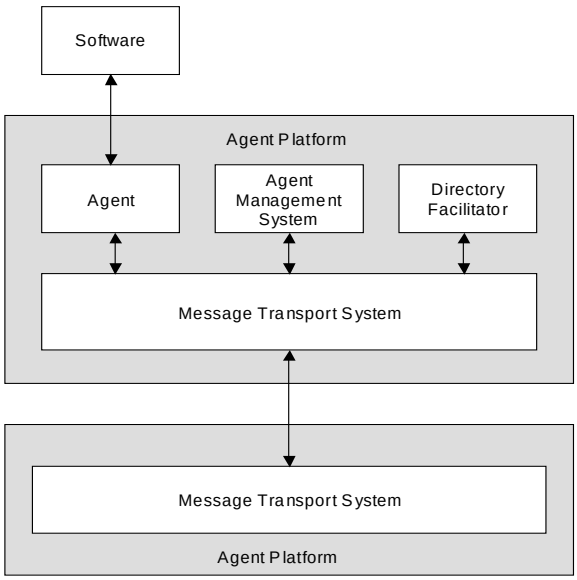


Fig. 1 Estructura de un sistema multiagente

Los componentes que forman parte de la *figura 1* son:

– Agente: unidad básica. Se podría definir como un programa que contiene y ofrece una serie de servicios.
– Directory Facilitator (DF): agente que proporciona un servicio de *páginas amarillas* dentro del sistema, es decir, conoce los diferentes servicios que ofrecen el resto de agentes de la plataforma. Los agentes se han de registrar en el DF para ofrecer sus servicios.

– Agent Management System (AMS): es el agente que controla el acceso y uso de la plataforma. Almacena las direcciones de los agentes, ofreciendo un servicio de *páginas blancas*.
– Message Transport System (MTS): facilita la comunicación entre los agentes de diferentes plataformas.
– Agent Platform (AP): proporciona la infraestructura básica para crear y ejecutar agentes.
– Software: cualquier programa accesible desde un agente.

2.2.3. Lenguaje de comunicación entre agentes (ACL)

Los agentes individualmente proporcionan una funcionalidad interesante, pero lo que les hace tan adecuados para ciertos sistemas es su capacidad de cooperar para resolver problemas. Para poder hacerlo, los agentes se han de comunicar entre sí, utilizando un lenguaje común: ACL *(Agent Communication Language)*. Para garantizar la homogeneidad y compatibilidad entre los diversos agentes, la FIPA determina que forma ha de tener un mensaje[7] y su utilización; para ello, esta organización elabora las *FIPA Specifications*[8] [specs05].

2.2.3.1. Componentes de un mensaje

Todo mensaje estará compuesto por una serie de campos (o slots) que son los siguientes:

Slot	Categoría a la que pertenece
Performative	Definición del tipo de mensaje
Sender	Participante de la comunicación
Receiver	Participante de la comunicación
Reply-to	Participante de la comunicación
Content	Contenido
Language	Descripción del contenido
Encoding	Descripción del contenido
Ontology	Descripción del contenido
Protocol	Control de la conversación
Conversation-id	Control de la conversación
Reply-with	Control de la conversación
In-reply-to	Control de la conversación
Reply-by	Control de la conversación

- Definición del tipo de mensaje: indica qué tipo de comunicación deseamos hacer.

o Accept-proposal: aceptamos una propuesta hecha anteriormente para realizar una acción
o Agree: aceptación de una acción
o Cancel: cancelación de una acción
o Cfp: para proponer una acción

[7] *Unidad básica de comunicación según la FIPA.*
[8] *Especificaciones presentadas por la FIPA, que forman el estándar para la creación de agentes y SMAs*

- Confirm: el emisor informa al receptor que una proposición es cierta, cuando el receptor lo dudaba
- Disconfirm:el emisor informa al receptor que una proposición es falsa, cuando el receptor pensaba que era cierta.
- Failure: indica que la acción pedida anteriormente ha fallado por algún motivo.
- Inform: el emisor informa al receptor que una acción se ha realizado correctamente.
- Not-understood: el emisor informa al receptor que no ha entendido una petición realizada anteriormente.
- Propose: acción de proponer la realización de una acción, dadas ciertas precondiciones.
- Query-if: acción de preguntar a otro agente si un hecho es cierto o no.
- Refuse: negación a la realización de una acción dada.
- Request: el emisor pide al receptor la realización de una acción.
- Subscribe: el emisor pide ser avisado en el momento que se cumpla una condición.

• Participante de la comunicación: son los identificadores de los agentes. Hay 3: sender (quien envía el mensaje), receiver (quien lo recibe), reply-to (a quien tiene que ir destinado el siguiente mensaje de la conversación).

• Contenido: existen cuatro tipos de contenidos predefinidos que se utilizan en función de las necesidades que se tengan.

- FIPA-CCL *(Constrant Choice Language)*: define una semántica que permite especificar predicados con restricciones.
- FIPA-SL *(Semantic Language)*: permite formar expresiones lógicas, intercambiar conocimientos de forma óptima y expresar acciones a realizar. Es el lenguaje más general de todos y puede ser aplicado a muchos dominios diferentes. Es el que ha sido utilizado en el SMA creado.
- FIPA-KIF *(Knowledge Interchange Format)*: permite expresar objetos como términos y proposiciones como sentencias.
- FIPA-RDF *(Resource Description Framework)*: permite expresar objetos, interacciones y acciones entre ellos.

• Descripción del contenido: permite que el agente que reciba el mensaje pueda identificar qué se le está enviando y en qué formato. Se definen 3 descriptores:

- Language: en qué está escrito el contenido, nosotros hemos utilizado FIPA-SL.
- Encoding: indica si se utiliza algún tipo de codificación especial.
- Ontology: es el campo más importante, ya que define la ontología utilizada para dar significado al contendido del mensaje.

• Control de la conversación: identifica el tipo de conversaciones que se mantienen. Sus campos son:

- Protocol: utilizado en el mensaje.
- Conversation-id: identificador de la conversación.
- Reply-with: identificador del mensaje que se utiliza para seguir los diferentes pasos de la conversación.
- In-reply-to: identifica una acción pasada a la cual este mensaje es la respuesta.
- Reply-by: marca de *time-out*: fecha/hora de caducidad del mensaje.

2.2.3.2. Protocolos de comunicación

Como hemos indicado anteriormente en *control de la conversación*, existe un campo que identifica el protocolo utilizado. Dicho protocolo es el que define una serie de reglas o pasos que se ha de seguir para desarrollar una conversación. Existen diferentes tipos dependiendo de su finalidad:

- FIPA Request: permite pedir la realización de acciones y devolver los resultados
- FIPA Query: se utiliza para hacer preguntas.
- FIPA Contract Net: es un protocolo de negociación (se proponen y se evalúan propuestas).
- FIPA Iterated Contract Net: variante de la anterior que permite la renegociación.
- FIPA Brokering: gestiona los servicios disponibles para los agentes.
- FIPA Recruiting: variante de la anterior.
- FIPA Subscribe: permite la notificación de hechos importantes.
- FIPA Propose: simplificación del contract net.

3. JADE

JADE[9] es una herramienta de programación que contiene un conjunto de librerías escritas en JAVA [Java05] para el desarrollo de Sistemas Multi-Agente [sma05]. Además de proporcionar las funciones de manejo de agentes a bajo nivel y las interficies que facilitan la programación, también nos presenta un entorno de ejecución donde los agentes podrán ejecutarse e interactuar [Isern et al., 2001].

Para la realización del proyecto hemos trabajado con la versión 3.1 que presenta las mejores prestaciones en su vertiente móvil que estudiaremos más adelante.

Comentaremos brevemente las especificaciones de JADE dejando para el lector la posibilidad de profundizar más en su funcionamiento consultando el *JADE Programmer's guide*, que encontraremos junto a mucha más información en la página oficial de JADE [guide05], o leyendo anteriores proyectos presentados por el GRUSMA.

3.1. Paquetes de JADE

JADE está compuesto por una serie de paquetes en Java. Los principales son los siguientes:

- *JADE.core* es el núcleo del sistema. Proporciona la clase JADE.core.Agent que es imprescindible para la creación de SMA. También contiene el paquete JADE.core.behaviour que incluye las clases de los comportamientos que ha de tener todo agente.

- *JADE.lang* contiene las clases específicas para soportar un lenguaje de comunicación entre agentes. Esta formado por el paquete java.lang.acl.

- *JADE.content* contiene las clases específicas para soportar un lenguaje de comunicación entre agentes (ACL). Está formado por los paquetes JADE.content.lang.acl y JADE.content.lang.sl. En éste mismo encontramos todas las clases necesarias para la realización e implementación de la ontología (JADE.content.onto).

- *JADE.domain* contiene todas las clases para representar la plataforma de agentes y los modelos del dominio, tales como entidades para la administración de agentes, lenguajes y ontologías (AMS, DF, ACC…).

- *JADE.proto* paquete que contiene los protocolos de interacción entre agentes FIPA.

- *JADE.tools* contiene algunas herramientas que facilitan el desarrollo de aplicaciones y la administración de la plataforma:

    - *Remote Management Agent* (RMA): es el agente que se encarga de la interfaz gráfica de JADE para la administración y control del sistema.
    - *Dummy Agent*: es una herramienta de monitorización y depuración, compuesta por una interfaz gráfica y un agente JADE. Permite la creación de mensajes ACL y enviarlos a otros agentes pudiendo visualizarlos.
    - *Sniffer*: agente que intercepta mensajes ACL y los visualiza, resulta muy útil para el seguimiento de una conversación.
    - *Introspector Agent*: agente que permite la monitorización del ciclo de vida de un agente.

[9] *Java Agent Development Enviroment*

- *SocketProxyAgent* agente que actúa como gateway bidireccional entre la plataforma JADE y una conexión TCP/IP.

- *JADE.gui* contiene herramientas para construir interficies gráficas

3.2. Componentes principales de JADE

Para JADE, los agentes residen en un entorno predefinido llamado plataforma. Cada plataforma está dividida en contenedores y cada uno de ellos contendrá agentes. Los contenedores podrían asemejarse al dominio de los agentes.

La plataforma se enciende en una máquina determinada, mientras que los agentes podrían ejecutarse en cualquier estación; esto nos ofrece un comportamiento distribuido del SMA que será la base para la incorporación del LEAP.

Tal como se define en la FIPA, en cada plataforma se inician dos agentes que tienen unas funciones vitales para el funcionamiento del sistema:

1. *DIRECTORY FACILITATOR (DF):* podría definirse como un servicio de "páginas amarillas", es decir, contendrá todos los servicios que proporcionan los agentes que se han dado de alta en la plataforma. Será de especial utilidad a la hora de requerir algún servicio, puesto que nos informará de quien puede proporcionárnoslo.

2. *AGENT MANAGEMENT SYSTEM (AMS):* se podría considerar como un servicio de "páginas blancas", donde se registran las direcciones de los agentes que se han dado de alta. Dado que JADE se fundamenta en entornos distribuidos, será básico para conocer la localización de los agentes.

Gracias a estos agentes la comunicación en la plataforma es muy sencilla de realizar y permite una gran flexibilidad y transparencia. Por ejemplo, si un agente A quiere enviar un mensaje a un agente B que desarrolla la tarea T, el agente A preguntará al DF qué agentes me pueden proporcionar dicho servicio T. Cuando se quiera conocer la dirección física de un agente determinado, preguntará al AMS. Para que este sistema funcione correctamente será necesario que, durante el proceso de inicialización del agente, éste informe al DF de qué servicios dispone y el tipo de agente qué es (para poder identificarlo); al hacerlo, AMS le dará una dirección física (dependiendo de dónde se ejecute), para que otros agentes se puedan poner en contacto con él.

3.3. Agentes y comportamientos (Behaviours)

Para poder crear agentes en JADE deberemos extender la clase JADE.core.Agent que ya está definida y deberemos rellenar los campos y las funciones básicas que nos proporciona la interficie. Dos métodos muy importantes que se han de implementar son setup() y takedown() que inicializan y finalizan el agente respectivamente [JADE Api].

Como inciso de interés comentaremos la existencia de la clase JADE.gui.GuiAgent, una extensión de la clase agente estándar, creada para facilitar la implementación de los agentes con GUI[10], que no ha sido utilizada en este proyecto dado que se fundamenta en la librería grafica *swing* y nosotros, debido a las limitaciones impuestas por LEAP y J2ME, hemos utilizado la librería *lcdui*[11].

[10] *Graphic user interface*
[11] *Liquid Cristal User Interface: librería gráfica de J2ME para la implementación de GUIs en móviles.*

Una vez hayamos creado el agente deberemos darle funcionalidad. Eso se consigue a través de la definición de los *comportamientos (behaviours)* que son los que controlan sus acciones. Los behaviours son los encargados de darle dinamismo al agente y que reaccione a estímulos (mensajes) externos.

3.3.1. Comportamientos SIMPLES

Son los comportamientos que no pueden tener hijos. Estos comportamientos derivan de la clase *JADE.core.behaviours.Behaviour*. Hay una serie de métodos que se pueden sobrecargar, como por ejemplo: action() (que engloba el conjunto de sentencias a ejecutar), done() (que devuelve un boolean que nos indica si el comportamiento ha finalizado) y el reset() (que reinicia el comportamiento). Hay tres clases de comportamientos simples:

- SimpleBehaviour: representa un comportamiento atómico que se ejecuta sin interrupciones. Podemos diferenciar entre:
  - OneShotBehaviour*:* sólo se ejecuta una vez.
  - CyclicBehaviour: el comportamiento se ejecuta cíclicamente.
  - TickerBehaviour: periódicamente se ejecuta un trozo de código indicado por el usuario.
  - WakerBehaviour: Oneshot task que se ejecuta tras producirse un time-out.
- ReceiverBehaviour: comportamiento que espera la recepción de un mensaje.
- SenderBehaviour: es equivalente al anterior pero desde el punto de vista de quien envía el mensaje.

3.3.2. Comportamientos COMPUESTOS

Son los comportamientos que pueden tener hijos, es decir, podemos asociar uno o más *subbehaviours* a un comportamiento complejo, que serán gestionados por un *scheduler* independiente del agente. Estos comportamientos derivan de la clase *JADE.core.behaviours.ComplexBehaviour*.
Los métodos que se pueden sobrecargar son onStart() y onEnd(). El primero en ejecutarse será el onStart(), que será dónde añadiremos los subbehaviours. Cuando finaliza la ejecución de este método se lanzarán todos los subbehaviours, y cuando todos hayan acabado se ejecutará el método onEnd(). Cabe destacar que en la antigua versión 2.61 estos métodos eran llamados *preAction()* y *postAction()* respectivamente, apunte que será de interés a aquellos más curtidos en la versión antigua.

Hay dos tipos de comportamientos complejos:

- SequentialBehaviour: ejecuta los hijos de manera secuencial.
- ParallelBehaviour: ejecuta los hijos según una política de Round Robin.
- FSMBehaviour: ejecuta los hijos según una máquina de estados finita.
- ReceiverlBehaviour: Comportamiento para recibir mensajes.

3.4. Lenguaje de Comunicación

El lenguaje FIPA-SL (Semantic Language) es un lenguaje formal con el cual se pretende poder resolver una gran variedad de problemas. En FIPA-SL se pueden formar expresiones lógicas,

intercambiar información entre agentes y expresar acciones a realizar. La notación utilizada es similar al Lisp: se definen un conjunto de predicados genéricos y cada uno de ellos tiene una serie de atributos y parámetros.

3.5. Ontología

JADE también nos permite crear ontologías. Éstas son muy importantes, ya que definen el vocabulario utilizado en nuestro SMA. Es por eso que antes de definir los agentes, hay que diseñar la ontología que se adapte a nuestras necesidades.
El elemento básico de una ontología es el frame. Éste a su vez se divide en slots, que pueden ser de diferentes tipos: SIMPLES (integer, string, boolean...) u OTROS FRAMES.
Para la creación de una ontología completa se han de tener 3 elementos en cuenta:

1. CONCEPTOS u OBJETOS (frames): define los elementos básicos de la ontología.
2. PREDICADOS: define una serie de propiedades o características que pueden cumplir los objetos. Ejemplo: “edad mayor de 18”, “que vivan en la ciudad de TGN”, …
3. ACCIONES: define las acciones que se podrán pedir a los agentes del sistema.

3.6. Protocolos de comunicación

Una vez definida e implementada la ontología, deberemos escoger el protocolo de comunicación entre agentes que más se adapte a nuestras necesidades. Para cada conversación, JADE distingue dos roles:

- El que inicia la conversación: INITIATOR
- El que la sigue: RESPONDER

JADE en su condición de plataforma que cumple las especificaciones de la FIPA, nos proporciona los elementos necesarios (*behaviours* y *performatives*) para definir los protocolos descritos por esta organización. Los protocolos entre los que podemos elegir son: FIPA-Request, FIPA-Propose, FIPA-Query, FIPA-Request-When, FIPA-Recruiting, FIPA-Brokering y FIPA-Contract-Net. Veamos una breve descripción de los protocolos más importantes que JADE implementa:

3.6.1. FIPA Request

Se utiliza cuando un agente quiere pedir la realización de una acción a otro agente. En la figura 2 podemos ver el protocolo, los cuadros blancos son el *initiator* y los grises el *responder*.

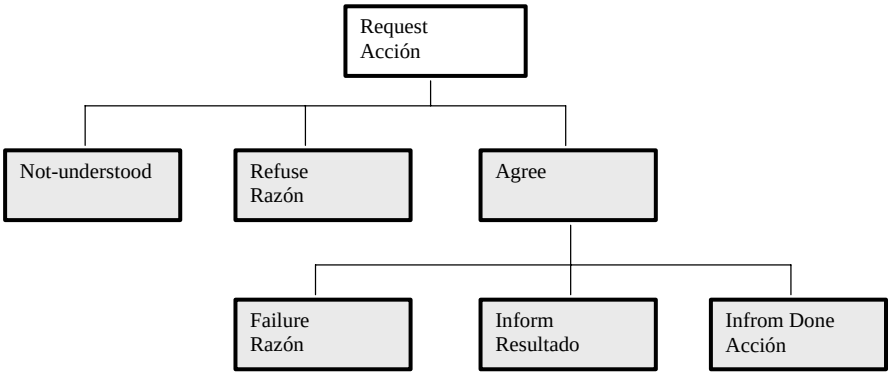


Fig. 2 Protocolo Fipa Request

3.6.2. FIPA Query

Se utiliza cuando un agente quiere pedir cualquier tipo de información a otro agente. En la figura 3 podemos ver el protocolo, los cuadros blancos son el *initiator* y los grises el *responder*.

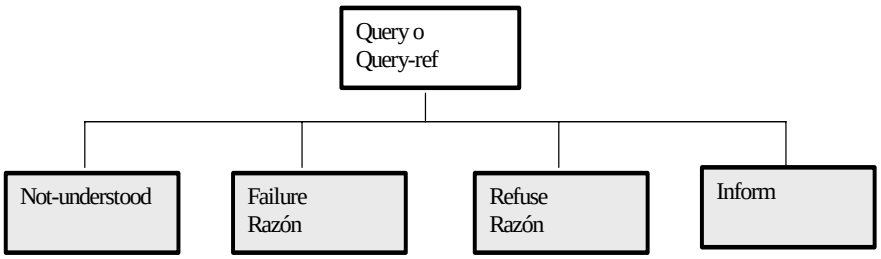


Fig. 3 Protocolo Fipa Query

3.6.3. FIPA Contract Net

Se utiliza cuando se quiere realizar una negociación entre varios agentes. En la figura 4 podemos ver el protocolo, los cuadros blancos son el *initiator* y los grises el *responder.*

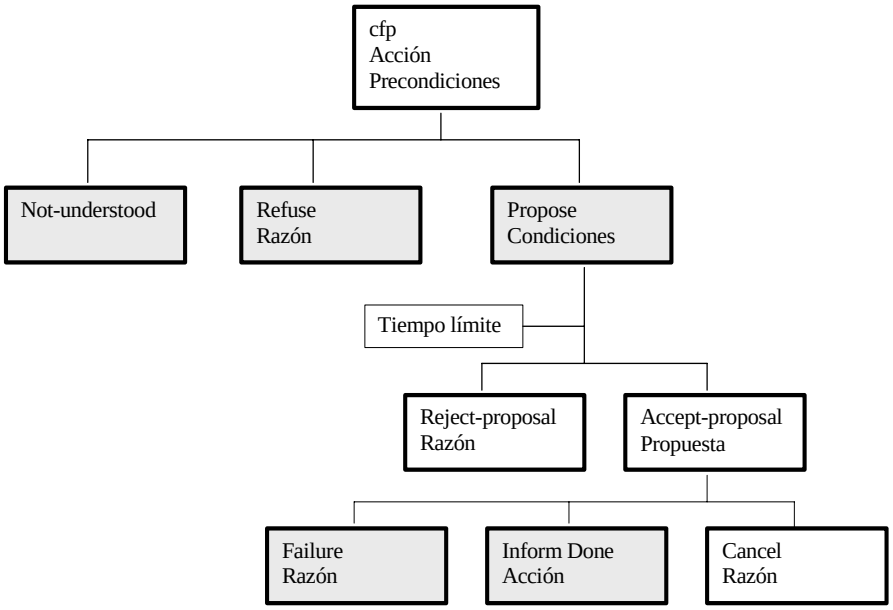


Fig. 4 Protocolo Contract Net

3.6.4. Clases proporcionadas por JADE

JADE proporciona diferentes clases para poder manipular los protocolos de comunicación establecidos según la FIPA:

- AchieveREInitiator / AchieveREResponder
- ContractNetInitiator / ContractNetResponder
- SubscriptionInitiator / SubscriptionResponder

3.7. Herramientas de JADE

JADE ofrece una interficie gráfica para la administración de la plataforma, así como herramientas para facilitar la depuración y testeo de las aplicaciones.

Para iniciar una sesión y crear los agentes, deberemos escribir la siguiente línea de comandos:

java JADE.Boot [options] –platform [lista de agentes]

En el campo de opciones podemos especificar que deseamos visualizar la interficie gráfica de JADE utilizando el parámetro *–gui*.

Para la lista de agentes debemos colocar una cadena de texto donde cada uno de ellos estará separado con espacios en blanco y tendrá el siguiente formato: *<NombreAgente>:ClaseAgenteJava*. En el caso que las clases las tengamos dentro de un package deberemos colocar la siguiente línea de texto: *<NombreAgente>:PackageClase.ClaseAgenteJava*. Cabe destacar que este formato sólo es valido para JADE en estado puro. Si utilizásemos JADE con LEAP, como de hecho haremos en el ejemplo final, el formato variaría ligeramente: cada agente seria separado por punto y coma (;), y tampoco podemos olvidar que en caso que las clases requieran parámetros para su ejecución, éstos serán introducidos usando paréntesis al final de cada clase. Veamos un rápido ejemplo:

<NombreAgente>:Package.ClaseAgente(param); <NombreAgente2>:Package.ClaseAgente(p1,p2)

De todas formas, existe la posibilidad de iniciar el entorno gráfico de JADE sin ningún agente y posteriormente con las herramientas que te ofrece ir cargando los agentes que se deseen y pudiendo visualizar su comportamiento, nosotros no obstante, por comodidad, los cargaremos todos de inicio, utilizando además un fichero BAT, que agilizará el procedimiento.

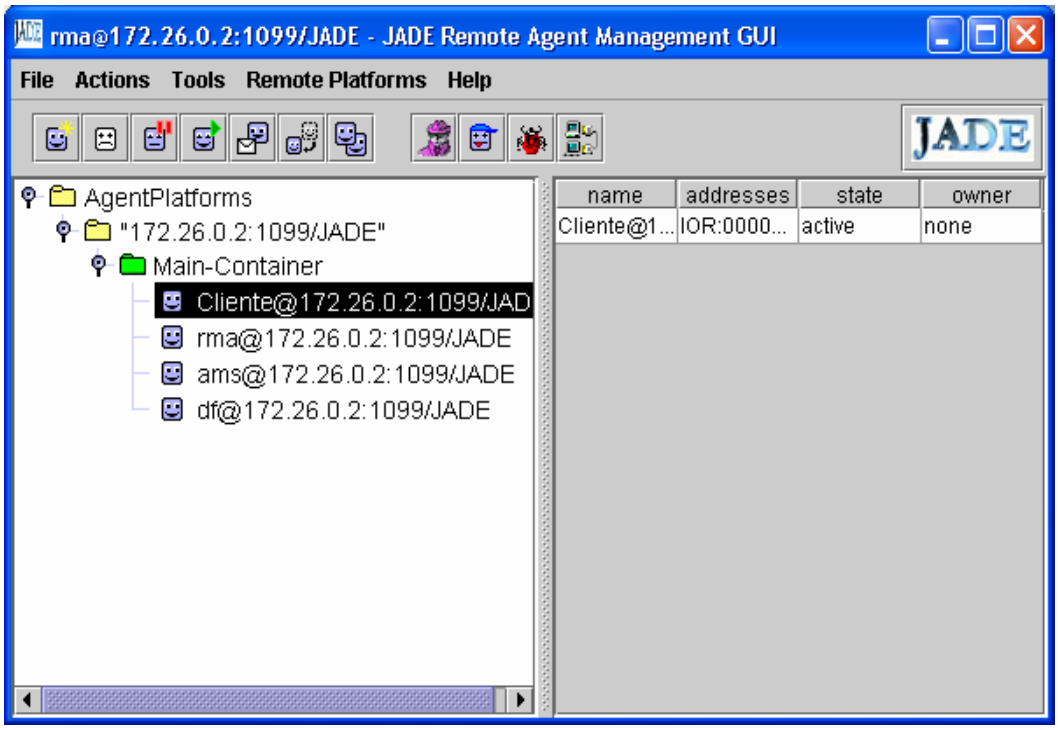


Fig. 5 Pantalla principal Jade

Cuando iniciamos el entorno gráfico, tal como se muestra en la figura 5, aunque no inicialicemos ningún agente, se crean algunos automáticamente, que son los agentes básicos de la plataforma, encargados de que todo funcione correctamente. Aunque ya han sido comentados en apartados anteriores, refresquemos un poco la memoria, para seguir correctamente el hilo de la explicación:

- *Agente RMA*: será el encargado de controlar la interficie gráfica de JADE.
- *Agente DF*: donde se subscribirán los servicios de los agentes. Es muy importante, ya que se utiliza para buscar agentes con unas características indicadas (páginas amarillas).
- *Agente AMS*: donde se guardan las direcciones de los agentes de la plataforma.

AGENTE SNIFFER: es una aplicación JAVA creada para rastrear el intercambio de mensajes entre los agentes JADE. Cuando lo activas se abre una ventana y sólo podrás observar que existe una caja llamada others. Para que el agente funcione correctamente y te muestre el intercambio de mensajes que se puede mostrar en la figura 6 debes indicarle qué agentes quieres visualizar.
Esta herramienta es muy útil para el testeo del correcto funcionamiento del paso de mensajes. Si haces doble click en las flechas se te abrirá una ventana como en la figura 6, donde tendrás toda la información necesaria sobre ese mensaje en concreto. Remarcar que otra posibilidad que nos ofrece es guardar los mensajes y posteriormente poderlos abrir.

Hay que destacar el serio inconveniente que presenta a los desarrolladores que usamos JADE-LEAP, y es que no nos permite hacer un seguimiento de los agentes situados fuera del *contenedor*[12] principal. De todos modos esto será tratado en el capitulo dedicado a LEAP.

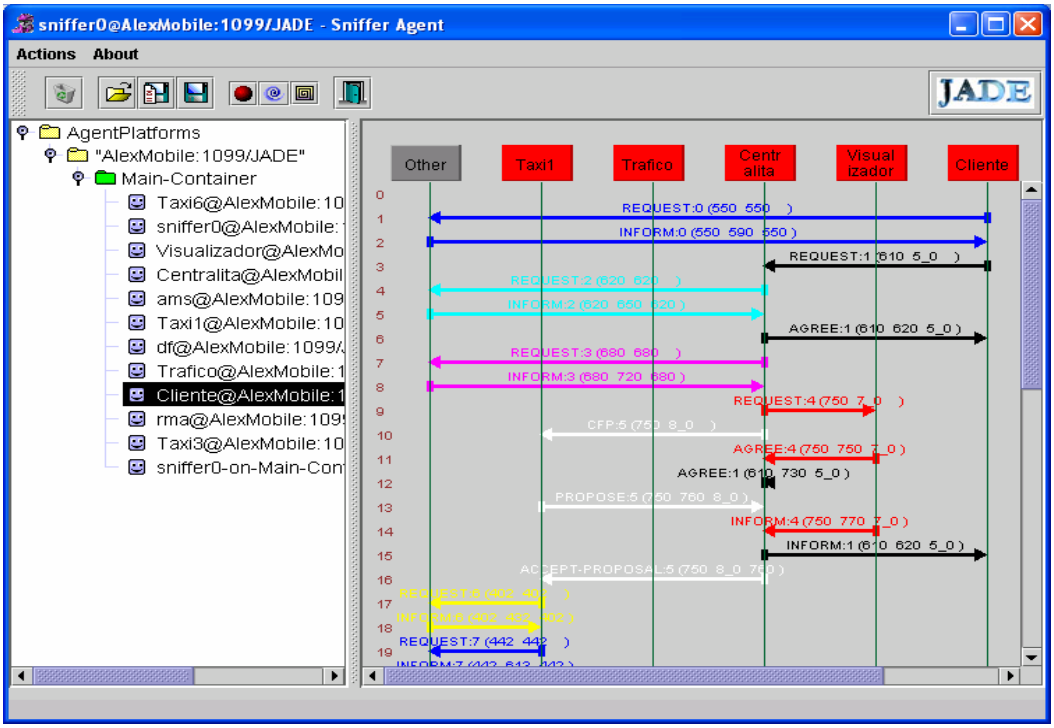


Fig. 6 Sniffer

[12] *En LEAP los agentes se sitúan en contenedores, donde cada contenedor representa una estación diferente, que puede ser un PC, una PDA, un móvil…*

AGENTE DUMMY *(Dummy-Agent)*: éste permite enviar mensajes ACL entre los agentes, recibirlos, inspeccionarlos, leer y salvarlos en un fichero (figura 7).

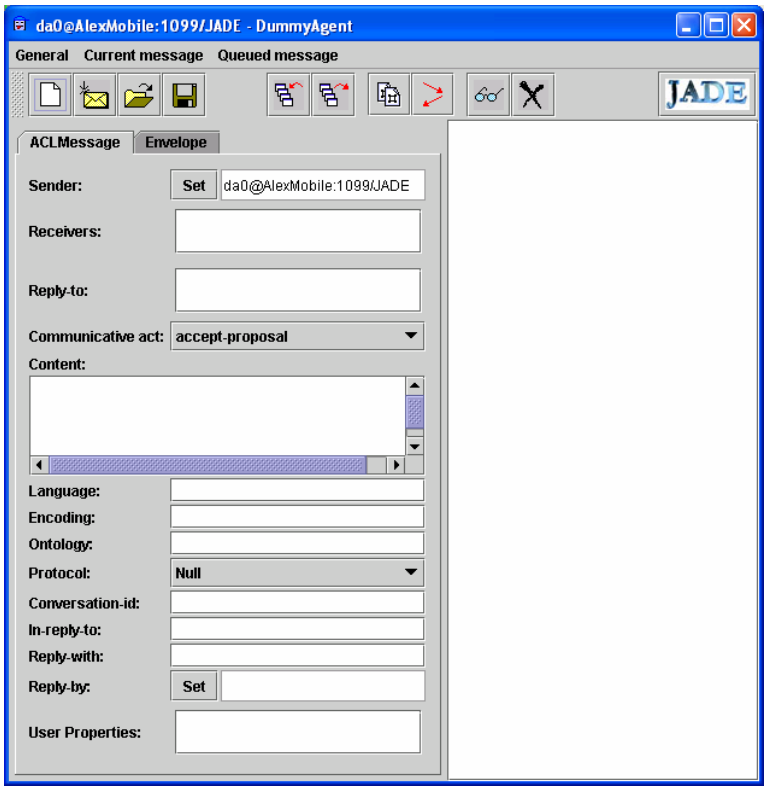


Fig. 7 Dummy Agent

DF: Otra herramienta muy útil para la realización de este tipo de sistemas, es la activación de la interficie del agente DF (menú *tools*), donde podemos observar los agentes que se han registrado y qué servicio ofrecen, tal como podemos observar en la figura 8.

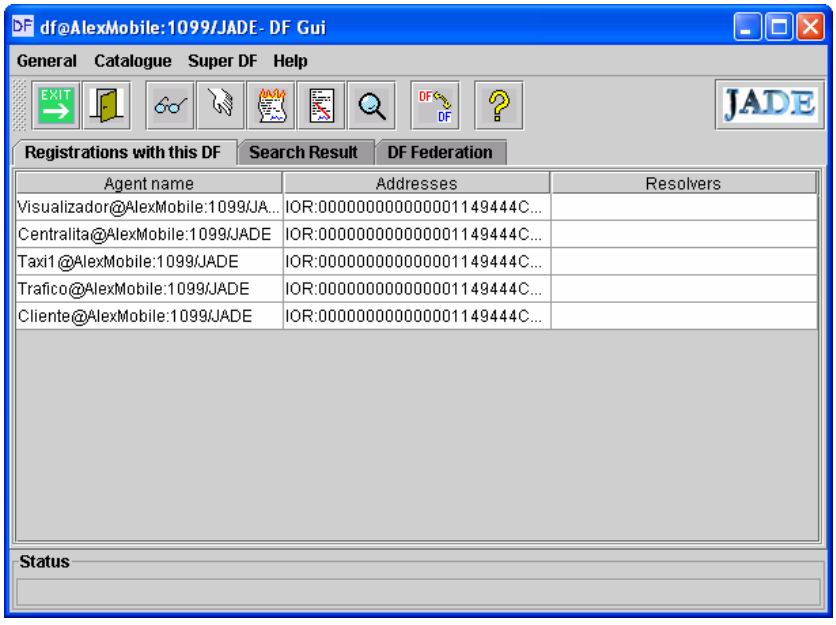


Fig. 8 Agente DF

INTROSPECTOR AGENT: nos permite monitorizar el ciclo de vida del agente (figura 9).

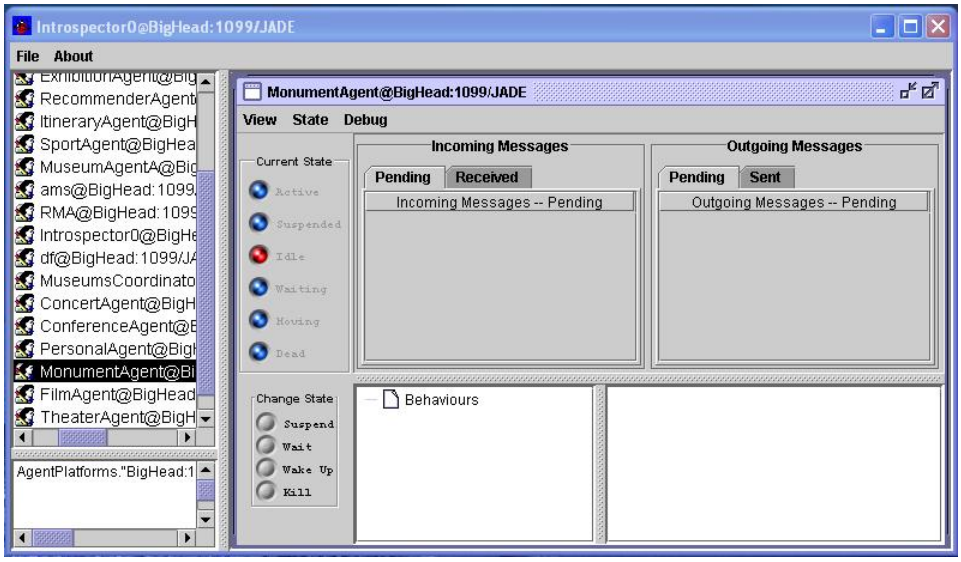


Fig. 9 Introspector

4. LEAP SOBRE JADE

LEAP (*Lightweight Extensible Agent Platform)* es un addon para JADE, que nos permite obtener un entorno de ejecución para agentes que cumplan las especificaciones FIPA y se ejecuten en dispositivos de recursos limitados, que soporten java.

Hasta la versión 2.1 LEAP fue desarrollado dentro del LEAP IST project [LEAP02]. El hecho de desarrollarse de forma externa al proyecto JADE provocaba ciertos problemas e inconvenientes, tanto a la hora de instalar el entorno JADE-LEAP[13], como a la hora de ejecutar los agentes en él (básicamente encontrábamos ciertas limitaciones que se convertían en dificultades añadidas). Las cosas cambiaron radicalmente desde la llegada de JADE3.0 que vino acompañada de su propio addon LEAP3.0. En nuestro proyecto utilizamos la versión 3.1 que ha resultado ser sencilla de instalar y muy cómoda de utilizar, facilitándonos el acceso al mundo de los dispositivos móviles.

La necesidad de LEAP viene dada por la incapacidad de JADE de ejecutarse correctamente en condiciones extremas de recursos, situación que nos encontramos claramente al trabajar en móviles o PDAs. JADE requiere un mínimo de JDK1.2 [14], problema añadido puesto que la mayoría de estos dispositivos solo soportan el llamado PersonalJava[15] o MIDP[16]. Por ultimo también hay que destacar la naturaleza de las conexiones de estos dispositivos: bajo ancho de banda, conexión intermitente, IPs dinámicas...Todo estos aspectos deben ser tratados de forma especial, es por ello que LEAP nació. Su base principal es crear un contenedor para cada dispositivo, donde sus agentes se ejecutarán. Estos contenedores, supeditados al contenedor principal (main container), formarán parte de una misma plataforma, que permitirá a todos los agentes trabajar conjuntamente.

Dicho esto, recordar que el addon LEAP puede ser descargado gratuitamente en la página oficial de JADE (http://JADE.cselt.it), en la cual también encontraremos un tutorial muy acertado sobre LEAP, que ayudará a profundizar al lector que se muestre interesado por esta plataforma.

4.1. Entorno de ejecución JADE-LEAP

LEAP, al ser combinado con JADE, substituye ciertas partes del kernel de JADE, creando un entorno modificado que identificamos como JADE-LEAP, que puede funcionar correctamente en dispositivos muy limitados desde el punto de vista de recursos. LEAP nos ofrece 3 versiones diferentes pensadas para cada uno de los 3 tipos posibles de entornos Java disponibles:

- j2se (Simple Edition): la indicada para ejecutarse en estaciones con jdk1.2 o superior
- pjava (PersonalJava) : idónea para dispositivos con PersonalJava como las PDAs.
- midp (Micro Edition): para dispositivos que soportan MIDP1.0 como teléfonos móviles.

Aunque internamente muy diferentes, las 3 versiones ofrecen la misma API a los programadores, sólo presentando ciertas diferencias las versiones j2se y pjava respecto a la midp. Cabe destacar que estas incompatibilidades están relacionadas con las clases de java no soportadas por MIDP. De todos modos, aunque realmente pocas, estas diferencias serán comentadas más adelante.

[13] *JADE powered by LEAP*
[14] *Java Development Kid. A dia de hoy se utiliza la versión 1.5.0*
[15] *Maquina virtual para PDAs. A día de hoy se utiliza su versión 1.2 (JeodeVM)*
[16] *Entorno facilitado por el J2ME de sun. Es el utilizado por los móviles. Las aplicaciones que ejecuta se llaman midlets.*

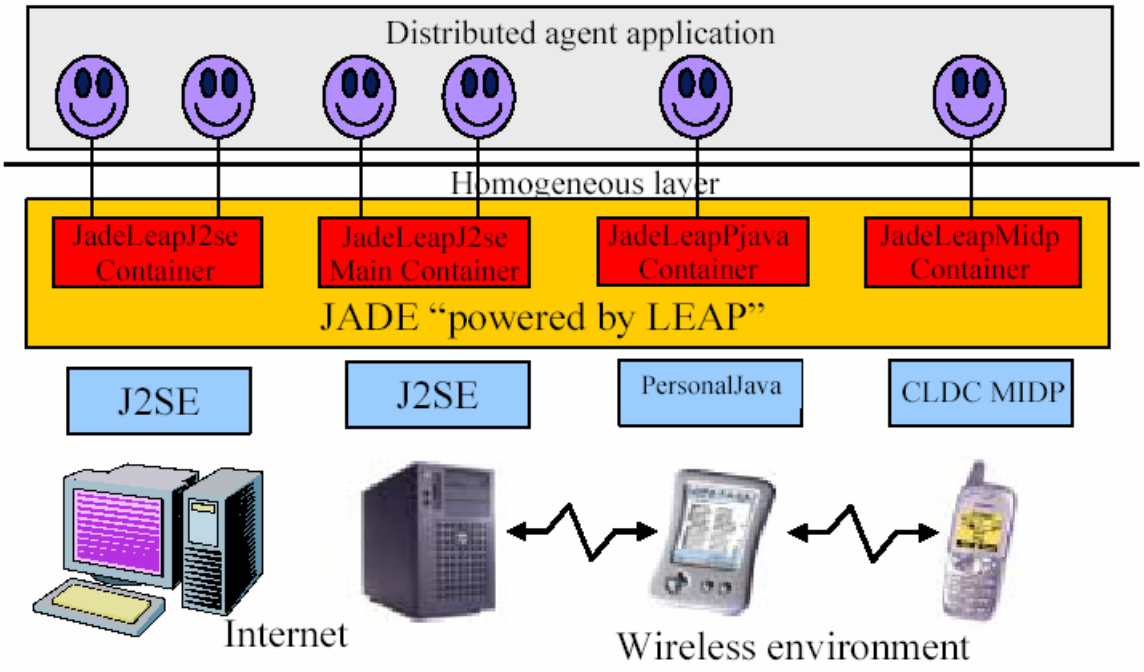


Fig. 10 Entorno ejecución JADE-LEAP

Desde el punto de vista del desarrollador de aplicaciones, JADE-LEAP para j2se es prácticamente igual a JADE, tanto en términos de APIs como de ejecución, solo teniendo unas ligeras diferencias que en nada afectan al código de la aplicación. No podemos dejar de comentar que, aunque sean tan similares, no podremos mezclar los contenedores de JADE con los contenedores de JADE-LEAP en una misma plataforma.

4.2. Ejecución de JADE-LEAP en estaciones (PC o Servidor)

Como ya se ha dicho, en este tipo de entornos la versión de LEAP idónea será la j2se. Sobre ella podremos actuar como si de JADE normal se tratase. Por lo tanto, asumiendo que tenemos las clases de JADE-LEAP en el classpath, la forma de arrancar la plataforma sería:

java JADE.Boot [options] [agents specification]

Donde la lista de agentes sería algo similar a:

local-name:package.agentclass(arg1,arg2);local-name2:package.agentclass2(arg1,arg2)

Mención especial para la separación por punto y coma, a diferencia de la usada en JADE (usando espacio en blanco).

Las opciones posibles serían:

- -main <true|false>: indica si la plataforma albergará el contenedor principal (por defecto true)
- *-container* : equivale a "– main false"
- -host <host adress> : indica la dirección del main container. Si es el main container lo ignora.
- -port <port-number> : indica el puerto donde escucha el main container. Si es el maincontainer lo ignora.

- -local-port <port-number> : indica el puerto donde este container escuchará.
- -gui : activa la RMA GUI
- -name <nom> : nombre que será asignado a la plataforma. Ignorado si es un contenedor normal.
- -container-name <nom> : nombre del container.
- -mtps : activa el MTPS
- -agents <lista de agentes separados por ; > : activa los agentes indicados.
- -conf : lee la configuración del fichero especificado (heredado de LEAP2.1).

4.3. Ejecución de JADE-LEAP en dispositivos con recursos limitados

Entrando de lleno en los dispositivos móviles, tanto teléfonos como PDAs, LEAP nos ofrece dos formas diferentes de ejecución, según lo críticas que sean las limitaciones de nuestro dispositivo:

STAND-ALONE

Es el modo de ejecución normal en el que un contenedor completo se ejecuta en el dispositivo. Es el modo que consume más recursos, solo indicado para el uso en PDAs con decentes prestaciones.

SPLIT

Este modo es el recomendado para agentes en teléfonos móviles. De hecho se ha comprobado que es el único modo que soporta la mayoría de los modelos, debido a las fuertes restricciones que dichos dispositivos imponen. En este modo, el contenedor se separa en dos partes llamadas FrontEnd (que se ejecuta en el dispositivo móvil) y BackEnd (que se ejecuta en un j2se host), unidas por un link a través de una conexión permanente. Las ventajas de este modelo son claras aprovechando que:

- El FrontEnd es claramente más sencillo que un contenedor completo
- La fase de bootstrap es más rápida.
- Hay menos envío de datos en la transmisión entre ambas partes.

En la figura 11 podemos ver un sencillo esquema de los dos modos de ejecución provistos por LEAP:

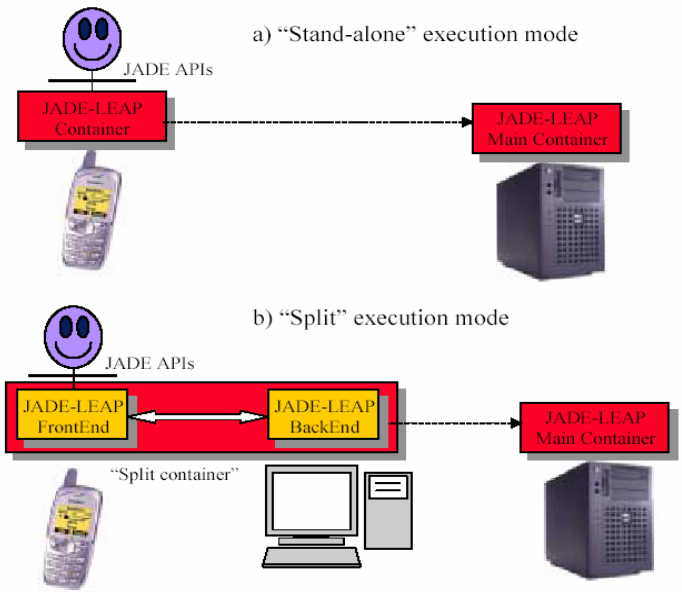


Fig. 11 Modos de ejecución

La ejecución en SPLIT, por tanto, permite que nuestro agente se ejecute ágilmente y que las esperas en los envíos de información al resto del SMA se reduzcan drásticamente. Naturalmente el programador no deberá preocuparse del modo de ejecución que presentará el container donde se ejecute su agente dado que las APIs son exactamente las mismas. No obstante sí tendrá que tener en cuenta que un Main container no puede ser Split. La movilidad de agentes y la clonación no son soportadas por un split container y además al lanzar un Split container, un j2se container tiene que estar activo en el host donde el BackEnd debe ser creado.

4.3.1. PDAs con PersonalJava

En este medio, la versión de LEAP idónea será la *pjava*.

Ejecución stand-alone:

java JADE.Boot [options] [agents specification]

Aplicaremos las mismas opciones que en el caso j2se, recordando que la opción *–gui* deberá ser excluida, dado que las herramientas de JADE requieren jdk1.2 para funcionar.

Ejecución split:

java JADE.MicroBoot [options] [agents specification]

Con las siguientes opciones disponibles:

- -host <host address> : indica el host donde se encuentra el BackEnd
- -port <port number> : indica el puerto donde escucha el BackEnd
- -agents <lista de agentes separados por ; > : activa los agentes indicados
- -exitwhenempty : el container acabará cuando no tenga más agentes ejecutándose.
- -conf : lee la configuración del fichero especificado.

4.3.2. Móviles con MIDP1.0

En este medio, la versión a utilizar de LEAP será la midp. Para conseguir introducir la plataforma LEAP en dispositivos MIDP contamos dos opciones:

1. Configuraremos un MIDlet Suite[17] con los siguientes midlets:

- JADE.Boot : carga este midlet para iniciar un stand-alone container
- JADE.MicroBoot : este iniciara un split container
- JADE.util.LEAP.Config : midlet para editar manualmente la configuración.
- JADE.util.LEAP.OutputViewer: midlet para hacer una sencilla depuración de las ejecuciones.

Las opciones son las mismas que en el caso del pjava, con la misma distinción para el *–host* y *-port* si estamos en modo Stand-alone o en modo Split.
Lo que sí debemos recordar en este caso es que estas opciones de configuración pueden ser introducidas de dos maneras:

- Como propiedades en el JAD del midlet.
- Utilizando el JADE.util.LEAP.Config midlet.

[17] *Conjunto de varios MIDlets que trabajan juntos.*

2. Configuraremos el MIDlet para que arranque una única clase implementada por nosotros que extenderá la clase MicroBoot. Desde dicha clase podremos lanzar nuestro agente mediante la llamada:

   MicroRuntime.startAgent(nombreagente, "AgentesTurismo.PersonalAgent", null);

   Para introducir las opciones de configuración tendremos dos maneras:

   - Como propiedades en el JAD del midlet.
   - Implementando el método: *protected void customize(properties p)*

Cabe destacar que extender la clase MicroBoot y utilizar el método customize, es la manera más cómoda de lanzar la ejecución y la que ha sido utilizada en la realización de nuestra aplicación.

4.4. Ejemplo final del entorno JADE-LEAP

Para finalizar este apartado dedicado a la ejecución del entorno JADE-LEAP, presentamos un último ejemplo representado por la figura 12, que incluye un Stand-alone container funcionando sobre PersonalJava en una PDA, un Split container funcionando sobre un MIDP móvil, y una estación con j2se que hace la función de contenedor principal. Como detalle de interés podemos ver que se muestran las propiedades de configuración para cada contenedor.

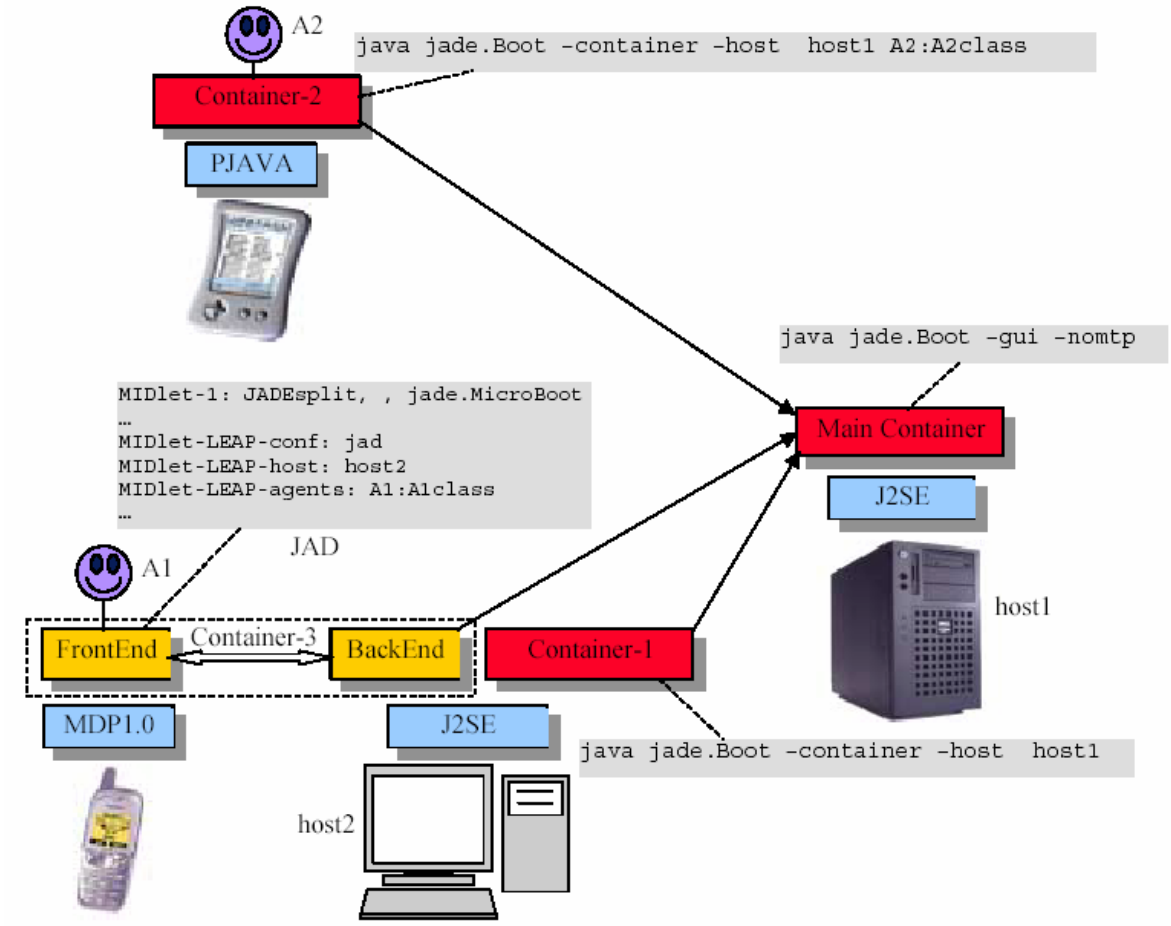


Fig. 12 Ejemplo final

4.5. Limitaciones en JADE-LEAP para pjava y midp

Como prometimos anteriormente, aquí presentamos una lista de las limitaciones de LEAP en sus dos versiones para dispositivos de mano, tanto PDAs como teléfonos móviles.

PJAVA

- Todas las herramientas de administración de JADE tienen GUIs basadas en Swing. Dado que pjava sólo soporta AWT, estas herramientas no pueden ser ejecutadas en un pjava container.
- No es posible usar el Sniffer sobre un agente ejecutándose en un *pjava stand-alone container*. No obstante, sí es posible hacer el sniff sobre un agente en un *pjava split container*.

MIDP

- Las mismas limitaciones que se aplican a pjava también se aplican a midp.
- No se puede realizar movilidad o clonación de agentes.
- Los reflective introspectors: *JADE.content.onto.ReflectiveIntrospector* y *JADE.content.onto.BCReflectiveIntrospector*, no son soportados por midp.
- El paquete *JADE.wrapper* y los métodos de la clase *JADE.core.Runtime* que se refieren a clases en ese paquete, no están disponibles en midp.

4.6. JADE-LEAP y la conexión sin cables (wireless)

Hemos hablado sobre la aplicación de LEAP en entornos distribuidos, formados por estaciones, PDAs y móviles, cada uno con sus propios agentes, y todos ellos trabajando juntos en una misma plataforma. Ahora nos es obligado abordar el problema de la conectividad, puesto que estos dispositivos, para poder comunicarse, deben estar conectados a través de algún medio. De esta parte JADE-LEAP no se ocupa. Este entorno de ejecución trabaja a más alto nivel, y simplemente establece conexiones TCP/IP entre los containers, sin preocuparse del medio por cómo estos se producen. Por lo tanto seremos nosotros los que deberemos facilitarle esa conexión.

Dada la naturaleza de las aplicaciones que funcionarán en un móvil, dicha conexión debe ser establecida mediante enlace sin cables: no podemos olvidar que por ejemplo en nuestro caso, el asistente personal estará ejecutándose en el móvil de un turista que andará libremente por la ciudad.

Las *wireless networks* (redes sin cables) son, por lo tanto, el tema que realmente nos interesa, y dentro este mundo podemos encontrar múltiples opciones: GPRS, UMTS, WLAN, Wi-fi, Bluetooth, infrarrojos...Estas quizás sean las opciones más famosas y por tanto las que cuentan con más apoyo. La mejor opción sería UMTS [UMTS05], aportándonos banda ancha y tarifa plana para móviles. No obstante dicha tecnología no esta totalmente extendida a día de hoy y por tanto queda aparcada para futuros proyectos. La siguiente opción sería GPRS, la conexión más común utilizada por los teléfonos móviles para acceder a Internet. Dado que ya está implantada a nivel mundial, y es accesible desde cualquier punto con la única ayuda de un proveedor y un dispositivo compatible, podríamos decir que es la opción a elegir. No obstante tampoco podemos olvidar el uso de Bluetooth que aunque presenta un alcance de entre 10 y 100 metros que lo hace totalmente inapropiado para su uso en un entorno real, presenta en contrapartida un uso gratuito del medio y por lo tanto lo convierte en una opción idónea para el debug de las aplicaciones. Queda descartado el uso de WLAN, Wi-fi e infrarrojos por ser tecnologías enfocadas a redes locales que ofrecerían un rendimiento similar al bluetooth pero sin su simplicidad y versatilidad.

4.6.1. Conexión Bluetooth entre PC y Nokia N-GAGE

La N-GAGE esta totalmente preparada para conseguir y mantener una conexión mediante Bluetooth, dado que la propia marca nos ofrece un software especifico (PC-Suite for Nokia N-GAGE) que sincroniza el móvil con el PC únicamente utilizando Bluetooth. De todas maneras conseguir la conexión sigue siendo tarea nuestra.

De la gran diversidad de software disponible para el establecimiento de enlaces bluetooth, nos decantaremos por la utilización de Plugfree 2.0 de Fujitsu, que siempre nos ha dado un optimo resultado a la hora de conectar diversos tipos de dispositivos.

Una vez el bluetooth dongle esta correctamente instalado en el PC mediante los drivers aportados por Fujitsu [Fujitsu05] y que podemos encontrar en:

http://support.fujitsu-siemens.de/drivercd/Files_gepackt/AllgPrgUtil/Bluetooth/WinXP_W2000/WinXP_W2000_Bluetooth_S6010_E7010_E7110_Rel-22-05-2002.zip

Procederemos a la instalación del software Nokia que incluirá la aplicación *mRouter*, indispensable para establecer correctamente la conexión, su función será detectar la presencia de un dispositivo bluetooth, en este caso la N-GAGE y establecer la conexión, para ello le indicaremos que busque dichos dispositivos en los puertos COM-bluetooth que el PC ofrece, como podemos ver en la figura 13:

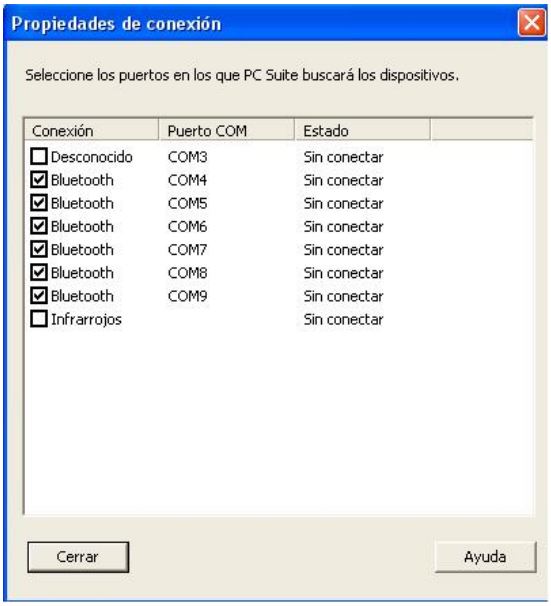


Fig. 13 mRouter

El paso ahora será usar plugfree para realizar la llamada a uno de los puertos COM-bluetooth que el mRouter controla para que este acabe de realizar la conexión. Procederemos a activar el módulo bluetooth de la N-GAGE y lanzaremos la aplicación plugfree, buscaremos los dispositivos presentes, seleccionamos la N-GAGE y tras obtener la lista de servicios disponibles, procedemos a seleccionar la conexión serie con el móvil, como podemos ver en la figura 14:

Fig. 14 Petición de conexión

El móvil nos avisará de la petición y tendremos que introducir un password para poder establecer la conexión. Una vez todo este validado, la conexión serie se establecerá y el mRouter mantendrá la conectividad entre PC y NGAGE. Podemos ver el estado actual del plugfree en la figura 15:

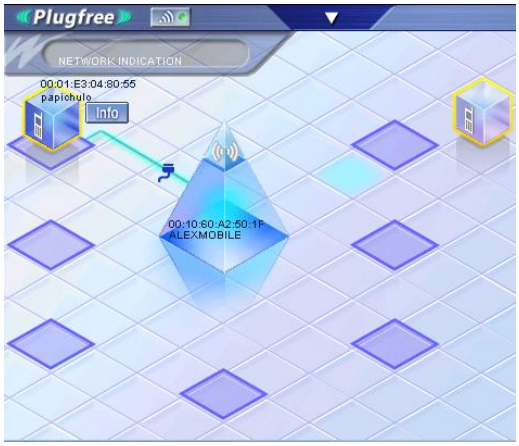


Fig. 15 Conexión establecida

Llegados a este punto tenemos en red a los dos elementos y por tanto podemos sincronizar la NGAGE con el PC. Para lanzar el SMA o cualquier otra aplicación que funcione de forma distribuida entre ambos dispositivos, deberemos crear en la NGAGE un nuevo punto de acceso asociado al Bluetooth, que será puesto por defecto en el menú de aplicaciones Java. Los parámetros de dicho punto de acceso son:

Nombre: BLUE
Portador de datos: GPRS
Nombre punto acceso (APN): ninguno
Nombre de usuario: ninguno
Petición contraseña: no
Contraseña: ninguno
Autentificación: normal
Dirección IP pasarela (gateway): 0.0.0.0

4.6.2. Conexión GPRS entre PC y Nokia N-GAGE

Debido a la naturaleza de los móviles y su conectividad mediante GPRS, el establecimiento de la conexión será sensiblemente diferente y es que en este caso, no crearemos el enlace a priori y luego lanzaremos las aplicaciones sino que será el propio móvil, el cual al lanzar la aplicación (agente en nuestro caso) establecerá la conexión "al vuelo", utilizando la configuración correcta.

Para ello, necesitamos un PC que sea accesible desde Internet y un determinado puerto abierto donde escuchará la plataforma JADE-LEAP. Hay que tener cuidado con routers trabajando en modo multi-puesto que realizan las funciones de firewall y por lo tanto no permitirán conexiones externas a menos que los configuremos correctamente. Naturalmente también tendremos que tener en cuenta cualquier otro software que haga funciones de cortafuegos en el PC.

Una vez el PC esta perfectamente configurado y estamos escuchando el puerto indicado, necesitamos configurar el móvil, que será el encargado de lanzar la conexión al mismo tiempo que ejecuta el agente.

Para que el teléfono haga su trabajo correctamente, deberemos crear un nuevo punto de acceso con los parámetros idóneos para establecer la conexión con Internet vía GPRS, para ello hemos obtenido dichos parámetros de la página web del grupo mms.es [mms.es05] que podemos ver en el cuadro representado por la figura 16:

	Amena	Movistar Plus	Movistar Activa	Vodafone
Nombre de conexión	Amena GPRS	Movistar GPRS	Movistar GPRS	Vodafone GPRS
Portador de datos	GPRS	GPRS	GPRS	GPRS
Nombre punto acceso	internet	movistar.es	p.movistar.es	airtelnet.es
Nombre de usuario	CLIENTE	MOVISTAR	MOVISTAR	wap@wap
Petición contraseña	No	No	No	No
Contraseña	AMENA	MOVISTAR	MOVISTAR	wap125
Autentificación	Normal	Normal	Segura	Normal
Dirección IP pasarela	0.0.0.0	0.0.0.0	0.0.0.0	0.0.0.0
Dirección IP teléfono	Dinámica	Dinámica	Dinámica	Dinámica
DNS primaria	213.143.33.8	194.179.1.100	194.179.1.100	212.73.32.3
DNS secundaria	213.143.32.20	194.179.1.101	194.179.1.101	212.73.32.67

Fig. 16 Parámetros acceso a Internet vía GPRS

Introducimos los parámetros de nuestro proveedor en el punto de acceso creado y desde el menú de aplicaciones Java seleccionamos dicho punto de acceso por defecto. Al lanzar las aplicaciones Java, en nuestro caso el agente personal, la aplicación en funcionamiento usará la conexión a Internet vía GPRS que acabamos de facilitar, estableciendo de esta manera la conexión sin cables con la plataforma JADE-LEAP que teníamos escuchando en un determinado puerto del PC situado en un lugar remoto. A partir de este momento la conectividad se mantendrá entre ambos dispositivos hasta que decidamos finalizar la ejecución en un extremo u otro.

5. J2ME

J2ME [J2ME05] es la línea de Java orientada a dispositivos con recursos limitados como móviles. Más específicamente, el conjunto de clases que oferta al programador sería la llamada MIDP (Mobile Information Device Profile), de la que se conocen dos versiones: 1.0 y 2.0.

MIDP esta diseñado para trabajar sobre CLDC (Connected Limited Device Configuration) en sus versiones 1.0 y 1.1

Lo expuesto aquí esta extraído del tutorial sobre MIDP escrito por F. Javier García Castellano [F.JavierGarcía05] y Luis Javier Herrera Maldonado [L.JHerrera05], ambos miembros del Departamento de Ciencias de la Computación e I.A (DECSAI) de la Universidad de Granada, así como de la zona asociada a la tecnología J2ME, que encontramos en la página de Sun Microsystems [Sun05]. Invitamos al lector interesado en esta tecnología a consultar dichas fuentes para ampliar los conocimientos que aquí mostramos.

5.1. Introducción

CDLC proporciona los aspectos básicos para ejecutar Java en pequeños dispositivos; no obstante, para la realización de una aplicación real, no nos sirve de mucho ya que su funcionalidad es más bien escasa. CLDC no tiene clases que nos permitan interaccionar con los usuarios, con el entorno de nuestra aplicación, con otras aplicaciones a través de la red o de poder acceder a dispositivos de almacenamiento. Esto es debido a que CLDC está concebido para que sea la base sobre la que se asienten aquellos perfiles que si nos van a proporcionar las funcionalidades que nuestras aplicaciones van a necesitar.

MIDP es el acrónimo de Perfil para dispositivos de información móvil (Mobile Information Device Profile) y nos proporciona un perfil que se apoya en CLDC y que nos va a proporcionar los paquetes y clases necesarios para el desarrollo de nuestras aplicaciones. MIDP está orientado principalmente a teléfonos móviles; no obstante, existe una implementación para PalmOS versión 3.5 y superiores. Además también se esta trabajando desde KObjects [KObjects05] con software que adaptaría su uso a PocketPC, por lo que podemos decir que MIDP es también utilizable en PDAs con estas características.

Se denomina Midlet a las aplicaciones Java realizadas usando la especificación de MIDP, es decir, un midlet será una aplicación que puede usar la funcionalidad aportadas por MIDP y por CLDC. Un midlet siempre estará compuesto, al menos, por una clase principal que hereda directamente de la clase *javax.microedition.midlet.MIDlet*

5.2. La API de MIDP

El API se compone de las clases y paquetes heredados de CLDC y de otra serie de paquetes situados en la jerarquía *javax.microedition* y podemos verlos a continuación:

- Paquete de Ciclo de vida de las Aplicaciones (javax.microedition.midlet): Este paquete permite a las aplicaciones MIDP (midlets) interactuar con el entorno, sobre el cual la aplicación se está ejecutando.

- Paquete de Interfaz (javax.microedition.lcdui): Este es la parte del API dedicada al interfaz de usuario (UI -User Interface). Proporciona un conjunto de características para la implementación de interfaces en MIDP.

- Paquete de Juegos (javax.microedition.lcdui.game): Este es la parte del API dedicada a juegos. Proporciona una serie de clases que permiten construir juegos ricos en contenidos para dispositivos móviles. No presente en MIDP 1.0 .

- Paquete de Red (javax.microedition.io): MIDP proporciona soporte de red basándose en CLDC.

- Paquete de Clave Pública (javax.microedition.pki): Certificados usados para autentificar información proveniente de conexiones seguras. No presente en MIDP 1.0 .

- Sonido:

    - javax.microedition.media: El API Media de MIDP 2.0 es un bloque directamente compatible con la especificación Mobile Media API (MMAPI). MMAPI extiende la funcionalidad de J2ME proporcionando audio, video y otras características multimedia. Es un paquete opcional, simple y ligero, que también permite acceder a los servicios multimedia nativos de nuestro dispositivo móvil (como la cámara de fotos de los móviles). No presente en MIDP 1.0.

    - javax.microedition.media.control:Este paquete define los tipos de control específicos que pueden ser usados en el reproductor (Player) del API Media. No presente en MIDP 1.0.

- Paquete de Persistencia (javax.microedition.rms): MIDP proporciona este mecanismo para que los midlets guarden persistentemente datos y posteriormente puedan recuperarlos.

- Paquetes principales:

    - java.lang: Las clases del lenguaje incluidas en el perfil provenientes de J2SE.

    - java.util: Las clases de utilidades incluidas en el perfil provenientes de J2SE.

5.3. El Midlet

Un midlet siempre heredará de la clase *javax.microedition.midlet.MIDlet*. Los métodos de esta clase permiten a nuestra aplicación crear, empezar, parar y destruir un Midlet, estos métodos son la interfaz del Midlet, que va a permitir a nuestro dispositivo poder manejar múltiples Midlets

Los Midlet necesitan ser empaquetados antes de ser instalados en el dispositivo de destino. Se utilizará un fichero .JAR(Java ARchive) donde se tendrá el Midlet principal y todas aquellas clases, imágenes u otros ficheros que nos puedan ser necesarios en tiempo de ejecución. También se incluirá en el .JAR información (en el fichero *manifest*) que le explique al dispositivo el contenido del fichero .JAR. Esta misma información también se incluye en el fichero .JAD (Java Application Descritor).

Las clases del Midlet que son empaquetadas en el .JAR, deben estar compiladas en .class (obviamente) y verificadas antes de su utilización en un dispositivo, para verificar que no realizan ninguna operación no permitida.

Un fichero .jar puede contener varios Midlets, a esto se le denomina *Midlet suite* y permite compartir recursos, lo cual es aconsejable para minimizar el uso de los recursos del dispositivo y para una mayor reutilización de los componentes.

5.4. Interfaz gráfico

Dentro de MIDP tenemos dos alternativas a la hora de realizar el interfaz gráfico de nuestros midlets, usar componentes del API de alto nivel para interfaces de usuario o el API de interfaces de usuario de bajo nivel.

Con el API de interfaz de usuario de alto nivel es el dispositivo (o más bien la máquina virtual del dispositivo) quien se encarga de colocar los componentes, las barras de desplazamiento, la navegación y características visuales como el color, las formas, los fuentes y los elementos a visualizar. Además tiene un sistema de entrada de alto nivel asociado.

Con el API de interfaz de usuario de bajo nivel la aplicación tiene un control mayor sobre la pantalla. El motivo de este API es para aquel tipo de aplicaciones que necesitan un control y precisión absoluto a la hora de dibujar elementos en pantalla, como son los juegos (que son el tipo de aplicaciones que copan la mayor parte del mercado del software para móviles). Un mayor control, significa también una mayor responsabilidad: hay que dibujar todo lo que aparece en pantalla e interpretar cualquier entrada del usuario. Cabe destacar que cualquier aplicación que utilice esta API verá comprometida su portabilidad, dado que trabajamos directamente con los recursos del móvil. Por ejemplo tenemos que saber el tamaño de la pantalla para no pintar más allá del límite.

La biblioteca de interfaz de usuario, tanto la de alto nivel como la de bajo nivel se encuentran en el paquete *javax.microedition.lcdui*.

La clase *javax.microedition.lcdio.Display* representa el controlador lógico de la pantalla del dispositivo, donde el Midlet colocará su interfaz. Es la clase responsable de controlar la pantalla y la interacción con el usuario. El objeto *Display* no se crea, se obtiene con el método *getDisplay()*.

Cada pantalla que el Midlet necesita dibujar usando componentes del intefaz gráfico (o, mejor dicho, ítems en el argot de MIDP) o usando el API de bajo nivel, es una pantalla derivada de la clase abstracta *Displayable*. Una clase *Displayable* no será visible hasta que no se use el método *setCurrent()* de la clase *Display*.

Displayable es la clase base para todos los interfaces de usuario de un Midlet, pero como es abstracta no la podemos utilizar. Las clases que realmente podemos utilizar para realizar interfaces de usuario son sus clases derivadas, cuya jerarquía se muestra en la figura 17.

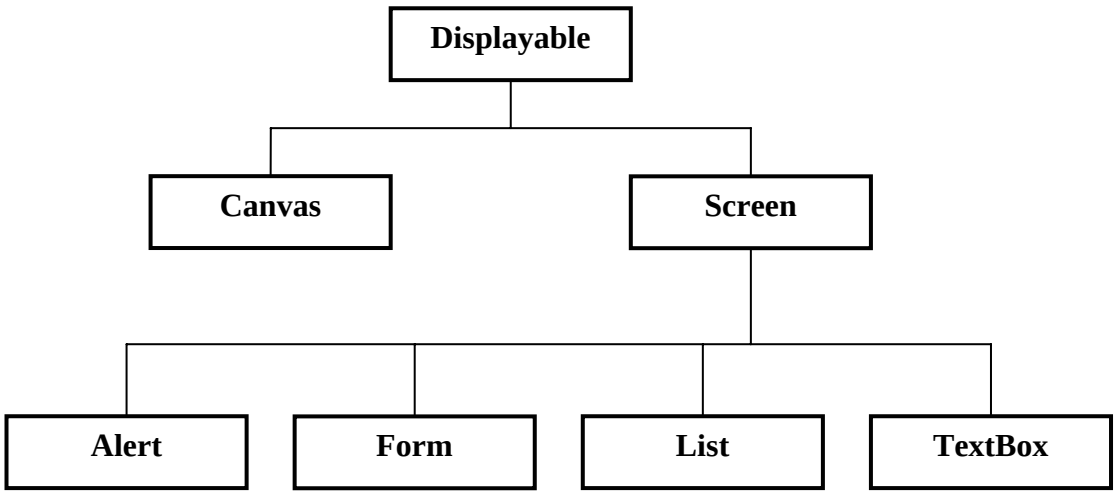


Fig. 17 Jerarquía de clases Displayable

Las dos clases que descienden de *Displayable*: *Screen* y *Canvas*, son también abstractas. Estas clases se corresponden con el API de alto nivel o el de bajo nivel del interfaz de usuario, respectivamente.

La clase *Canvas* es una pantalla en la cual el Midlet puede dibujar directamente. A la hora de crear un interfaz de bajo nivel para nuestro Midlet tendremos que crear una subclase de *Canvas* y redefinir el método *paint()*, para dibujar directamente en la pantalla del dispositivo.

La clase *Screen* es la clase base que usaremos en nuestro API de alto nivel del interfaz de usuario. No tenemos que heredar de ella, a diferencia de *Canvas,* y extiende la clase *Displayable* para que muestre un título de aplicación y un mensaje deslizante (ticker). La subclase más utilizada de *Screen* es *Form*, donde podremos colocar nuestros ítems. Aunque del objeto *Display* sólo puede haber una instancia, del objeto *Screen* puede haber muchas, aunque sólo se puede mostrar una pantalla cada vez.

La clase *Displayable* define los métodos: *addCommand*, *removeCommand* y *setCommandListener* que heredan todas sus subclases y que nos va a permitir tratar los eventos que se produzcan.

5.4.1. Interfaz gráfico de alto nivel

Screen es la clase base para utilizar el API de alto nivel del interfaz de usuario. Podemos agrupar las subclases *Screen* en dos tipos: un primer tipo encapsula completamente los ítems del interfaz de usuario siguiendo una estructura predefinida, dentro de este tipo tenemos *List*, *TextBox* y *Alert*, este tipo no permite que se añadan o quiten otro tipo de componentes. En el segundo tipo tenemos la clase *Form* que permite a las aplicaciones añadir o quitar ítems de la pantalla, tal y como desee el programador del Midlet, no obstante, la clase *Form* no permite encapsular completamente sus componentes como las clases del otro tipo de *Screen*.

Clase Form

Form como hemos visto es un contenedor de ítems del interfaz gráfico. Todos los ítems que se pueden añadir a *Form* descienden de la clase *Item*, la cual contiene las subclases:

- *ImageItem*: Clase que permite colocar imágenes en el interfaz gráfico.
- *StringItem*: Ítem para colocar una cadena en el interfaz.
- *TextField*: Un campo de entrada de texto de una sola línea
- *DateField*: Una versión de *TextField* especializada en la introducción de fechas.
- *ChoiceGroup*: Un componente que proporciona un conjunto de elecciones exclusivas o no, que puede funcionar como una colección de checkboxes o de radio buttons.
- *Gauge*: Un ítem para ver el progreso de una operación o para seleccionar un valor de un rango continuo de valores.

Clase Alert

La clase *Alert* hereda de la clase *Screen* y nos sirve para mostrar un mensaje (y opcionalmente una imagen) al usuario. Es como un pop-up o un MessageDialog. Al igual que la clase *Form* los cambios se reflejan automáticamente en pantalla (siempre que el Midlet esté Activo).

Clase List

La clase proporciona una lista de elementos de forma parecida al ítem *ChoiceGroup* pero no para integrarlo en un *Form* sino para ser usado directamente en el *Display*. Esta clase implementa la interfaz *Choice* por lo que su uso es análogo a la clase *ChoiceGroup* que también implementa dicho interfaz.

Clase TextBox

Nos permite editar un campo de texto, normalmente más de una línea y fuera de un *Form* a diferencia del ítem *TextField* que, normalmente, sólo nos permite editar una línea y dentro de un *Form*.

5.4.2. Interfaz gráfico de bajo nivel

En el campo de aplicaciones en Java para teléfonos móviles, los juegos son las aplicaciones que copan el mercado. Con los componentes de la API de alto nivel poco se puede hacer, nos hace falta acceder con total libertad a la pantalla del dispositivo y la interacción del usuario. No podemos dejar al dispositivo que elija donde colocar las imágenes de nuestro juego, por ejemplo.

A diferencia del API de alto nivel aquí tendremos muchas menos clases con las que trabajar, pero habrá que cuidar más los detalles, hasta nivel de píxel.

Hay que tener en cuenta que se puede pasar del API de bajo nivel a alto nivel y viceversa, pero lo que no se puede es mostrar las dos a la vez.

En el API de bajo nivel nos tenemos que encargar nosotros de manejar y procesar todas las entradas del usuario. Por lo tanto la gestión de comandos se realizará de forma manual tanto para el tratamiento de las teclas, como para el tratamiento del dispositivo apuntador, si este existiese.

Clase Canvas

La clase abstracta *Canvas* es como un folio en blanco en el cual el Midlet puede dibujar directamente. A la hora de crear un interfaz de bajo nivel para nuestro Midlet tendremos que crear una subclase que herede de *Canvas* y redefinir el método *paint()*, para dibujar directamente en la pantalla del dispositivo lo que queramos que aparezca. Este método es llamado con un sólo argumento, que es objeto de otra clase de bajo nivel, la clase *Graphics*. Esta clase nos proporciona métodos que nos van a permitir dibujar líneas, rectángulos y arcos; rellenar zonas con un color, y mostrar texto en la pantalla del dispositivo. La aplicación no tiene que ocuparse de llamar al método *paint()*, de ello se encarga el dispositivo.

Clase Graphics

La clase *Graphics* nos va a permitir, como hemos dicho, realizar operaciones para dibujar en la pantalla del dispositivo. Obsérvese que no tendremos que crearla, ya que el método *paint()* es llamado por el dispositivo. Veamos la lista de elementos que puede dibujar la clase Graphics:

1. Primitivas gráficas: líneas, rectángulos y arcos
2. Fuentes: Strings, caracteres…
3. Imágenes

5.5. Persistencia de datos

Prácticamente, en toda aplicación que se desarrolle, por muy pequeña que sea, vamos a tener la necesidad de almacenar datos. Siempre habrá cierta información que nos hará falta conservar entre distintas ejecuciones de una aplicación.

En los dispositivos móviles, tenemos poca capacidad de almacenamiento y carecemos de una conexión de red rápida y barata para acceder a los datos. Además, la forma de guardar información varía de dispositivo en dispositivo. MIDP nos proporciona una interfaz de programación independiente de cada dispositivo y que nos va a permitir tener almacenados datos en nuestro dispositivo móvil para nuestras aplicaciones independientemente de que se estén ejecutando en ese momento.

El paquete de java MIDP que se ocupa del almacenamiento persistente de datos es *javax.microedition.rms*[18].

Para que una aplicación pueda almacenar datos en el dispositivo móvil, debe crear un almacén de registros (RecordStore). Es una base de datos especial donde la unidad de información son los registros (Records). Una vez añadimos un registro a un almacén de registros, a éste se le asigna un identificador único (idrecord). Los almacenes de registros pueden ser compartidos por los MIDlets dentro de una misma Suite de Midlets, pero un MIDlet no puede acceder nunca a un almacén que no haya sido creado por un MIDlet que no esté dentro de su Suite.

Cada almacén de registros tiene un identificador que es una cadena de 1-32 caracteres Unicode, así pues, la combinación (nombre del almacén, MIDlet Suite) identifican unívocamente a un almacén de registros. Un MIDlet perteneciente a una Suite, puede acceder por tanto a cualquier almacén de datos de cualquiera de los otros MIDlets de su propia suite, pero sin embargo, no puede obtener ningún tipo de información de los almacenes de registros de cualquier otra Suite. Igualmente cabe notar que ninguna aplicación en un dispositivo móvil que no sea de Java tampoco puede acceder a los datos almacenados por los MIDlets de J2ME.

Un almacén de registros tiene cero o más registros, donde cada registro es un array de bytes con un identificador único asociado que permite identificarlo unívocamente dentro del almacén. Los identificadores se les asignan a los registros empezando por 1 y dando al siguiente registro insertado un identificador igual a 1 + el identificador dado al último registro añadido anteriormente. El borrado de los identificadores no es relevante en la secuencia dada a la inserción de registros

5.6. Conexión a redes

Debido a las dificultades para proveer soporte para funciones de red a nivel de configuración, por la variedad en los dispositivos, el CLDC delega esta parte del API a los perfiles. Para realizar esta delegación de forma satisfactoria, el CLDC ofrece un marco general de trabajo en red, conocido como el GCF (*Generic Connection Framework*). El GCF está compuesto básicamente por una serie de interfaces de conexión, junto con una clase "Conector" que es usada para establecer las diferentes conexiones. Todo esto, está dentro del paquete *javax.microedition.io*.

Las interfaces del paquete *javax.microedition.io* son:

- *Connection* – Una conexión básica que sólo puede ser abierta y cerrada
- *ContentConnection* – Un flujo (stream) de conexión que proporciona acceso a datos web

[18] *RMS: Record Management System*

- *DatagramConnection* – Una conexión para manejar comunicaciones orientadas a paquetes
- *InputConnection* – Una conexión de entrada para las comunicaciones del dispositivo
- *OutputConnection* – Una conexión de salida para las comunicaciones del dispositivo
- *StreamConnection* – Una conexión en ambas direcciones para las comunicaciones del dispositivo
- *StreamConnectionNotifier* – Una conexión especial para notificaciones, que es usada para esperar que se establezca una conexión

La clase fundamental para la conexión a redes es la clase "Connector". Siempre que queramos establecer una conexión, sea del tipo que sea, realizaremos una llamada al método estático *open()* de esta clase que nos devolverá un objeto que implementa la interfaz "Connection". El método *open()* tiene la siguiente sintaxis:

public static Connection open (String name)

Donde "name" es la cadena de conexión que tiene el siguiente formato:

protocolo: dirección; parámetros

- protocolo: indica si vamos a realizar una conexión de tipo http, socket o datagram
- dirección: indica la dirección de red del destino
- parámetros: es una lista de parámetros asociados a la conexión.

Por ejemplo podemos hacer:

```
Connector.open(“socket://172.26.0.4:3333");
```

Una vez establecida la conexión podremos enviar y recibir información siguiendo los métodos habituales ofrecidos por la API.

6. NOKIA N-GAGE

Siguiendo con nuestra descripción de las herramientas claves de este proyecto, dedicaremos este capítulo a describir el dispositivo móvil sobre el que desarrollaremos nuestro asistente personal, interficie del turista con el SMA que hemos implementado.

Esta importancia que le damos al dispositivo se explica porque dependemos increíblemente de la tecnología que éste nos aporte, para poder dotar al aplicativo de unas u otras funcionalidades, así como obtener dichas funciones de forma más o menos complicada. Por lo tanto, dejamos claro que la portabilidad a otros móviles dependerá en fuerte medida de que estos cumplan una serie de requisitos mínimos, sobre los cuales la aplicación se sustenta, que comentaremos de inmediato.

6.1. Características técnicas

La información referente a las características que Nokia [Nokia05] ha introducido en su N-GAGE pueden ser consultadas de forma extensa tanto en el ForumNokia [ForumNokia05] con la página web asociada: http://www.forum.nokia.com/main.html, como de la lectura de la web de NewLC [Newlc05] en concreto: http://www.newlc.com/article.php3?id_article=25.

De dichas fuentes podemos extraer el cuadro mostrado en la figura 18, que nos ayudará a remarcar los requerimientos por los cuales decidimos lanzar nuestra aplicación sobre este móvil en concreto.

Hardware specification	
Size	133.7x69.7x19.7mm
Weight	135g
Processor	ARM925@104Mhz
Hardware Accelerators	None
Internal storage	3.4MB
External storage	SD/MMC (64MB card included)
Screen Type	TFT
Screen Resolution	176x208 pixels
Screen Size	35*41.5mm
píxel Size	0.2mm
Colors	4096
Max Framerate	30fps
Audio	PCM 8/16kHz, 8bits, mono or stereo MP3 and AAC player, AAC encoder (TBC), FM radio
Keyboard	9-way round-cursor
Camera Resolution	None
OS Specification	
Symbian Release	6.1
GUI Platform	Series 60
Java Support	J2ME CLDC 1.0 / MIDP 1.0 .Nokia UI API, Wireless Messaging API, Mobile Media API
Communications	
GPRS	Rx/Tx: 2+2 or 3+1 (up to 40.2kbps (Rx) or 26.8kbps (Tx) with CS-2 coding)
HSCSD	Rx: up to 43.2kbps
IrDA	No
Bluetooth	1.1

Fig. 18 Características técnicas nokia N-GAGE

De dichas características, el punto clave es la conexión GPRS sin el cual el SMA no podría funcionar. De igual forma valoramos muy positivamente la conexión Bluetooth que nos permite tener un medio excelente para realizar el debug de la aplicación.

La memoria disponible del dispositivo es un punto a destacar. La aplicación ocupa del orden de 1MB, espacio que el móvil destino tiene que garantizar. Además la N-GAGE utiliza tarjetas de expansión de memoria (MMC) que serán de mucha utilidad en nuestro caso.

Las características del entorno Java las expondremos en el apartado siguiente, pero ya destacamos que son clave. Un dispositivo que no soporte J2ME, queda descartado para manejar nuestra aplicación.

Como parámetros determinantes a la portabilidad tenemos la especificación del sistema operativo. La N-GAGE trabaja con Symbian 6.1 [Symbian05] dentro de la especificación Serie60, la cual define un tamaño de pantalla y una resolución. Nuestra aplicación esta desarrollada bajo el SDK[19] Serie60 de Nokia para Symbian 6.1. Esto significa que dicha aplicación funcionará en móviles que cumplan las especificaciones dadas por Nokia para la Serie60 y que trabajen con Symbian 6.1.

Podemos concluir que, aunque a priori parezcan muchas restricciones, el móvil utilizado es relativamente sencillo de obtener y está sobradamente superado por los móviles actuales. Por ejemplo a día de hoy, dentro de la Serie60, podemos encontrar el Nokia 6630 con sistema operativo Symbian 8.0a y prestaciones muy por encima de las ofrecidas por la N-GAGE.

6.2. Soporte J2ME

En este apartado nos centraremos en las posibilidades que nos ofrece la máquina virtual java integrada en la N-GAGE. De la lectura preliminar de las prestaciones ofrecidas por el móvil, podemos saber que trabaja con MIDP 1.0 y CLDC1.0 que son las especificaciones más bajas posibles en cuanto a J2ME. Saber esto ya nos define sensiblemente nuestro radio de acción, puesto que contaremos con unas librerías muy reducidas y, por tanto, muchas dificultades a la hora de implementar las funcionalidades deseadas al aplicativo. Sin ir más lejos, ya sabemos que usando MIDP1.0+CLDC1.0 no tendremos posibilidad de utilizar floats, puesto que no son soportados en esta versión.

No obstante, necesitamos saber algo más para conocer las posibilidades reales del J2ME de la N-GAGE. Para ello utilizaremos un midlet llamado: *Sysinfo*, creado por Joerg Pleumann [Pleumann05] y descargado de la página de DeviceDB [devicedb05] enmarcada dentro del grupo de desarrolladores de software libre asociados a KObjects [KObjects05].

Sysinfo tiene como cometido realizar un chequeo de todas las capacidades que ofrece el entorno MIDP del dispositivo en cuestión. En detalle, intenta establecer una conexión de red a los protocolos que están disponibles, comprueba las limitaciones de las clases incorporadas y hace algún benchmarking en el RMS (Record Management System) y en el dispositivo en general. Finalmente los resultados son mostrados por pantalla.

Una vez descrito su funcionamiento, vamos a mostrar los resultados obtenidos por dicho midlet en nuestro móvil:

[19] *Software development Kit*

Propiedades:

CLDC-1.0
MIDP-1.0
NumColors=4096
CanvasSize=176x144

Protocolos:

http=true
https=false
socket=true
serversocket=false
datagram=false
comm=false
file=false

De estos resultados podemos extraer que nuestra aplicación solo podrá tener acceso al protocolo de *http* y solo estará capacitado para abrir un socket como cliente, nunca como servidor. Además podemos ver que el protocolo *file* esta desactivado, por lo tanto nuestra aplicación no tendrá acceso al sistema de ficheros, lo que significa que solo podrá acceder a los ficheros presentes en el JAR de la aplicación.

Con todos estos datos en la mano, podemos concluir que el soporte J2ME de la Nokia N-GAGE resulta bastante pobre en conjunto, limitándonos mucho a la hora de implementar el asistente personal.

Antes de cerrar este apartado debemos destacar las librerías java adicionales a las aportadas por J2ME, que son ofrecidas por Nokia, y que dan al programador un mejor control del dispositivo para desarrolladores de software centrados en juegos para móvil. Dichas librerías ofrecen un mejor acceso a los sonidos, un canvas mejorado, acceso al modulo de vibración y, en general, extras que el móvil puede ofrecer y que las librerías J2ME, al estar orientadas a un ámbito más genérico, no contemplan o solo contemplan en parte.

6.3. Soporte Visual C++

El dispositivo, de la mano de su sistema operativo Symbian 6.1 y sus librerías C++ asociadas, nos ofrece un completo elenco de clases disponibles, que en el caso que nos ocupa, serán de importante utilidad. Dichas librerías tienen acceso completo al dispositivo, permitiéndonos suplir con garantías las funcionalidades que no se pueden obtener mediante el limitado MIDP que viene incorporado en el móvil.

Es muy de destacar la presencia de la *C/POSIX Standard Library* que nos permite utilizar todas las funciones habituales del lenguaje C clásico, además de poder utilizar las librerías C++ asociadas a Symbian.

7. SMA TURIST@

7.1. Descripción general del aplicativo

El objetivo de nuestro sistema es ofrecer al turista un agente personal en forma de aplicación atractiva y cómoda de utilizar, que sea capaz de ayudarle en su recorrido por la ciudad, avisándole de actividades y puntos de interés para visitar a medida que vaya descubriéndola. El sistema atenderá sus peticiones en cuanto a tipos de actividades ofrecidas y dará la oportunidad al turista de puntuar las actividades realizadas. Todo ello dando un trato personalizado al turista, dado que el asistente tendrá en cuenta sus gustos y preferencias.

Con la ayuda del sistema de localización y un elenco de mapas de la ciudad, el asistente personal permitirá al usuario conocer su posición en cualquier momento, ayudándole a orientarse y mostrando de igual forma puntos de interés que se adapten a su perfil.

El asistente personal no trabaja solo, sino que se apoya en una serie de agentes ejecutándose en uno o varios servidores situados en algún lugar remoto. Dichos agentes están especializados en ciertas tareas: gestión de peticiones, gestión de recomendaciones, gestión de monumentos…y ayudan al asistente personal a ofrecer un eficiente servicio al turista.

Esta forma distribuida de trabajar se aprovecha de las tremendas ventajas que nos proporciona la implementación de un SMA para optimizar de forma sustancial todo el proceso que conlleva las demandas de información respecto a servicios ofrecidos por una ciudad.

El sistema viene a ofrecer un servicio de calidad al turista, dándole un trato personal y continuo. Complementándose así con el resto de puntos informativos dedicados al turismo y buscando la máxima satisfacción de aquellos que visitan la ciudad.

7.2. Descripción general del funcionamiento

En líneas generales, comentaremos los casos significativos del funcionamiento del SMA, de esta forma daremos una primera aproximación al lector de cómo trabaja nuestro aplicativo.

- Búsqueda explicita de actividades turísticas: Por ejemplo, el turista pregunta al asistente personal los monumentos y museos disponibles en la ciudad. El asistente envía la petición a un agente intermedio (Agente Broker) que reenviará dos peticiones, una para el agente encargado de monumentos y otro para el agente encargado de museos. Dichos agentes buscarán en sus respectivas bases de datos los monumentos o museos, según sea el caso, que se ajustan a los requerimientos del turista, sea por fechas de visita o por temática o precio del mismo. Una vez tengan los resultados, serán enviados al agente intermedio, que los juntará en un resultado global y los devolverá al asistente personal, resultados que a su vez mostrará al turista en la pantalla del móvil.

- Localización en la ciudad: El turista pide al asistente personal por su posición actual en la ciudad. El asistente utiliza el sistema de localización para obtener la posición y pregunta al agente encargado de recomendaciones si en la zona actual hay actividades de interés que se adapten al perfil del turista que tenemos en ese lugar. El agente recomendador consulta su base de datos adaptándose a los gustos del turista y envía al asistente los resultados, que serán mostrados al usuario junto con un mapa de la ciudad que indicará su posición.

- Recomendaciones: El turista solicita al asistente personal una lista de actividades que se ajusten a sus gustos personales. El asistente informa al agente recomendador de su identificador de usuario, y éste devuelve las actividades que se adaptan a los intereses del turista. También podemos destacar que el proceso de recomendación se realizará automáticamente cuando una nueva actividad se introduzca en el sistema o cuando un turista entre en una zona con actividades que puedan ser de su interés.

- Votaciones: El turista puntúa una actividad realizada en función de la satisfacción obtenida. Dicha valoración ayudará tanto para conocer el éxito de una determinada actividad, como para profundizar en los gustos del usuario.

- Agenda: El sistema ofrece al turista una agenda para poder almacenar las actividades que podemos considerar "favoritas". De esta manera facilitamos el acceso a la información asociada a dichas actividades por parte del turista, evitando esperas y gasto de dinero.

- Perfil de usuario: El sistema mantiene un perfil asociado a cada usuario para poder conocer sus gustos e intereses, facilitando un trato personalizado a cada turista. El perfil va actualizándose a medida que el turista interacciona con el asistente, adaptándonos así a sus preferencias y necesidades actuales.

7.3. Justificación de la utilización de un SMA

Las principales ventajas que nos proporciona el hecho de utilizar un Sistema Multi-Agente para este caso son:

- MODULARIDAD: El sistema ha de resolver un problema complejo compuesto de una gran cantidad de datos de diferentes ámbitos. Dicho problema se puede descomponer en tareas que pueden realizar diferentes actores (que son responsabilidad de diferentes personas) y así obtener la modularidad y mantener la independencia de los diferentes actores, que pueden actuar por sí solos según su propia conveniencia.

- EFICIENCIA: permite que los diferentes agentes se puedan ejecutar paralelamente (en diferentes máquinas) o concurrentemente (en la misma máquina), mejorando el tiempo de cálculo.

- FLEXIBILIDAD: Se pueden incorporar y eliminar fácilmente los diferentes agentes del sistema. Esto permite que el problema sea escalable, es decir, se pueden ir incorporando más agentes o clientes sin tener que modificar el código.

- COOPERACIÓN: La cooperación que se consigue con este tipo de sistemas hace que se puedan solventar problemas con más complejidad. En nuestro sistema se ve claramente dicha propiedad en el momento en que el agente intermedio separa una tarea compleja en varias tareas simples y las reparte entre un conjunto de agentes especializados. En futuras actualizaciones se podrían incluir nuevas técnicas de negociación entre agentes.

7.4. Ámbito de trabajo

Como ya hemos comentado, la función principal de nuestra aplicación es ofrecer información referente a actividades culturales y de ocio al turista que visita una ciudad. En nuestro caso trabajaremos con la ciudad de Tarragona, pero no existe ningún impedimento para que la aplicación trabaje con actividades de cualquier otro lugar.

Seguidamente, mostraremos los diferentes tipos de actividad que nuestro sistema soporta en la actualidad. Cada tipo está encargado a un agente especializado, por lo tanto, para poder trabajar con más tipos de actividad simplemente tendríamos que añadir más agentes. El lector se dará cuenta así de lo idóneo del uso de un SMA para este tipo de aplicación.

- Museos: Como por ejemplo el museo de arqueología de Tarragona.
- Monumentos: Como sería el acueducto romano
- Itinerarios: Itinerarios turísticos por la ciudad, como por ejemplo un itinerario por las playas.
- Exhibiciones: Exposiciones temporales de tipo artístico o religioso por ejemplo.
- Conferencias: Conferencias que se realizan puntualmente.
- Cine: Sesiones de cine puntuales donde se muestran películas de interés.
- Deportes: Encuentros deportivos como partidos de fútbol o básquet.
- Teatro: Representaciones teatrales en la ciudad.
- Conciertos: Conciertos que se realizan puntualmente.

7.5. Arquitectura del sistema

En el sistema tendremos cuatro tipos de agentes diferentes. Recordar que al estar trabajando en un entorno distribuido los elementos se pueden repartir entre varias estaciones. En nuestro caso el Agente Personal (o Asistente Personal) se encuentra en un móvil, mientras que el resto de agentes se encuentran en un ordenador convencional. En un caso más real, con un volumen de datos y peticiones sensiblemente superior seguramente sería más interesante situar los agentes en servidores independientes.

Los tipos de agentes que el sistema utiliza para su funcionamiento son los que detallamos a continuación:

- Agente Personal: El asistente personal que representa al turista a la hora de obtener recomendaciones y realizar búsquedas. Cada usuario tendrá su propio asistente personal.
- Agente Recomendador: Agente que realiza la tarea de recomendar y gestionar el perfil de los usuarios. Posee una base de datos con los perfiles de los usuarios y las actividades disponibles.
- Agente Broker: Agente que ofrece la posibilidad de realizar consultas de actividades.
- Agentes Especializados: Conjunto de agentes que representan los diferentes tipos de actividades presentes en la ciudad. Cada agente mantiene una base de datos propia con los detalles de sus actividades asociadas. Tenemos los siguientes tipos definidos:

    - Agente Monumentos
    - Agente Museos
    - Agente Itinerarios
    - Agente Exhibiciones
    - Agente Conferencias
    - Agente Conciertos

- Agente Deportes
- Agente Teatros
- Agente Películas

Exceptuando al Agente Personal, todos los demás agentes registrarán sus servicios en el Directory Facilitator (DF).Esto es debido a que, en cualquier momento, el sistema puede necesitar el trabajo de uno u otro agente, y en dicho caso hay que poder localizarlos con la máxima celeridad.

En la figura 19 podemos ver una visión global de la arquitectura del sistema que nos ayudará a entender las relaciones existentes entre los agentes del sistema y como se coordinan entre ellos. Mostramos, de forma general, los tipos de mensaje que se intercambian entre ellos y las bases de datos de cada agente.

Las bases de datos locales, asociadas a cada agente, son totalmente independientes entre si, lo cual nos permite trabajar en un entorno totalmente distribuido, y que modificaciones que efectuemos sobre un agente determinado no presenten ningún tipo de efecto sobre el resto, facilitando así la escalabilidad del sistema y el proceso de actualización y mantenimiento del mismo.

Una vez visto el esquema que muestra la arquitectura del SMA, comentaremos en detalle las funcionalidades que presenta cada tipo de agente para conseguir una mayor comprensión del sistema por parte del lector.

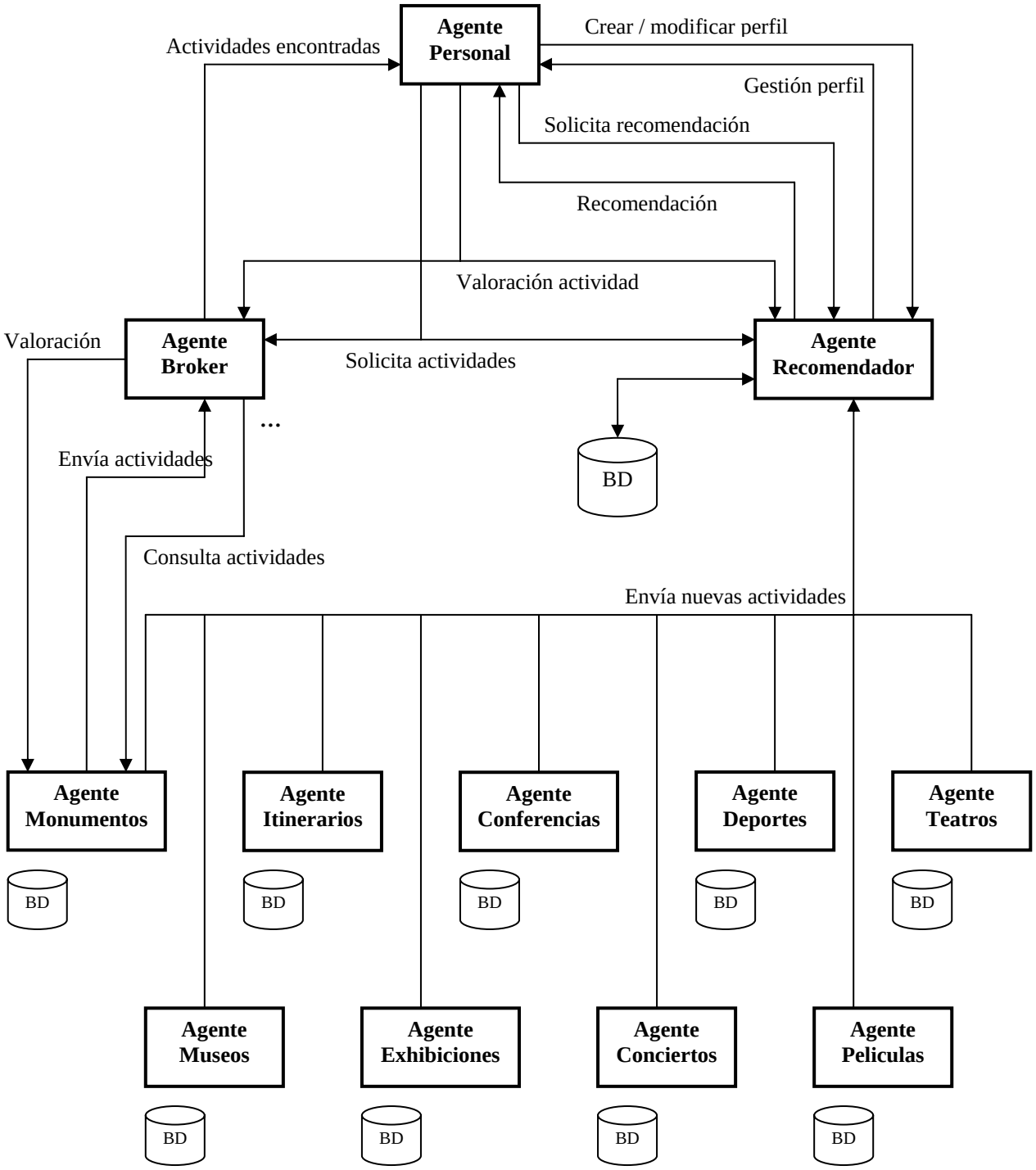


Fig. 19 Arquitectura del sistema

Notas sobre el esquema:

- El agente recomendador y cada agente de actividad tienen su propia base de datos local, totalmente independiente del resto.
- Los mensajes “Envía actividades”, “Consulta actividades” y “Valoración”; tienen lugar entre el agente Broker y cada uno de los agentes de actividades (agentes especializados).

7.6. Funcionalidades de los diferentes tipos de agente

Una vez tenemos una visión en conjunto de los agentes y sus interacciones es el momento de comentar dichos agentes en detalle:

7.6.1. Agente Personal

Representa un usuario. Tiene por objetivo realizar la función de interficie entre el SMA y el usuario. Su función es comunicarse con el agente Broker para realizar búsquedas de actividades que cumplan ciertos requisitos, y de igual forma, comunicarse con el agente Recomendador para obtener recomendaciones respecto actividades turísticas que puedan ser del interés del usuario.

Inicialmente busca en el DF un agente Recomendador y un agente Broker para poder empezar a trabajar. Si ha sido activado por primera vez, el agente personal mostrará un pequeño formulario por la pantalla del móvil. Dicho formulario representará el perfil inicial del turista, que a medida que vaya utilizando la aplicación irá modificándose y adaptándose a sus gustos personales.

Una vez dentro del menú principal, el turista podrá pedir recomendaciones, buscar actividades, ver, gestionar o valorar sus actividades favoritas (almacenadas en lo que llamamos la Agenda del turista), ver y modificar su perfil y conocer su posición actual en la ciudad, visualizando un mapa donde podrá orientarse.

La información de cada actividad mostrada al usuario incluye: nombre, tipo, lugar donde se desarrolla, horarios y fechas, precio, descuentos, valoración de los turistas, comentarios adicionales, visualización de su posición en el mapa correspondiente y foto asociada a la actividad.

7.6.2. Agente de Actividad

Llamamos agentes de actividad, o agentes especializados, al grupo de agentes presentes en el sistema que almacenan información sobre las actividades que la ciudad ofrece al turista. Los agentes de actividad pueden ser de monumentos, itinerarios, exhibiciones, conferencias, conciertos, deportes, teatros o películas.

Estos agentes contienen toda la información de la actividad que se encuentre dentro de su ámbito y se comunican con los agentes Broker y Recomendador para poder gestionar eficientemente las peticiones que realice el turista.

Inicialmente un agente de actividad busca en el DF un agente recomendador y le envía el conjunto de actividades que tiene en su base de datos local. Posteriormente avisará al recomendador cada vez que se añada una nueva actividad con el objetivo de mantener la coherencia entre datos. Así mismo el agente de actividad contestará a las peticiones del Broker, realizando búsquedas en su base de datos con unas ciertas restricciones.

7.6.3. Agente Broker

El agente Broker devuelve la información solicitada por los agentes personales al realizar una búsqueda de actividades turísticas. Ejerce de negociador en la búsqueda de la información requerida, separando un problema complejo en diversas tareas más simples que reparte entre los agentes de actividad. De forma similar, ejerce de intermediario en el envío de valoraciones por parte del usuario. El agente Broker realiza las tareas de Proxy para los agentes especializados modularizando el sistema al máximo y por lo tanto facilitando la eficiencia del SMA, además de ayudar a la escalabilidad del mismo.

En la figura 20 mostramos un pequeño diagrama donde pueden verse las interacciones del agente Broker con el resto de agentes del sistema:

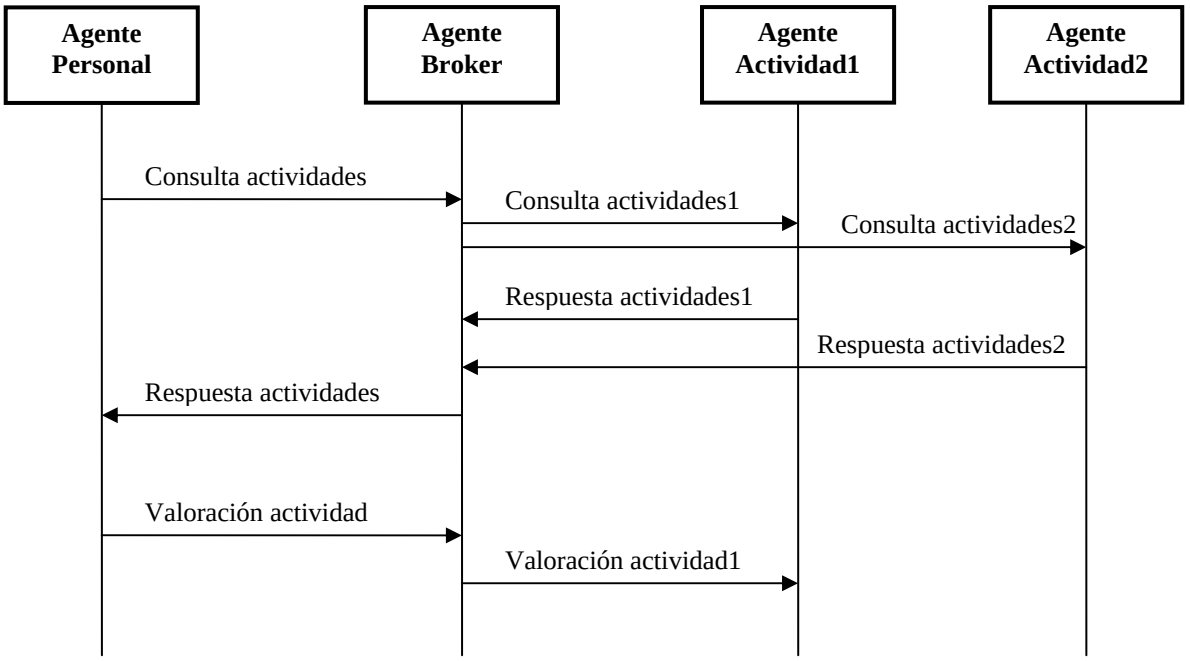


Fig. 20 Interacción del agente Broker con el resto de agentes

7.6.4. Agente Recomendador

Es el encargado de gestionar los perfiles de los usuarios y recomendar actividades que se ajusten a dichos perfiles. El agente recomendador utiliza la información sobre las búsquedas realizadas y las valoraciones proporcionadas por los mismos usuarios para actualizar dinámicamente el perfil de cada usuario. Dicho agente, almacena en su base de datos, tanto información de usuarios como de actividades para poder realizar estas tareas que tiene asignadas.

En la figura 21 podemos ver un diagrama donde quedan reflejadas las interacciones del recomendador con el resto de agentes del sistema:

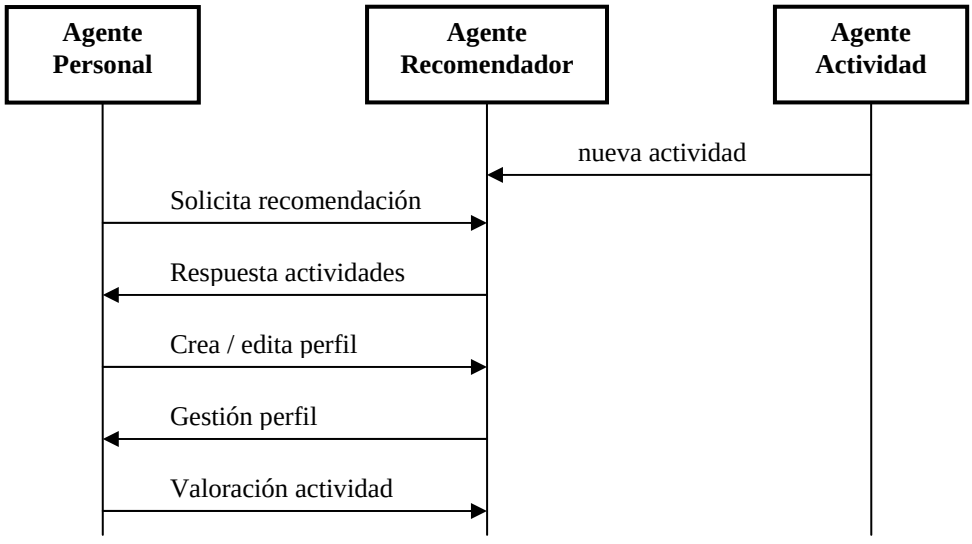


Fig. 21 Interacción del agente Recomendador con el resto de agentes

7.7. Comunicación entre agentes

El acto comunicativo en un SMA se basa en el envío de mensajes entre agentes. Para poderse comunicar correctamente, los agentes necesitan un protocolo, una ontología en común y un lenguaje que todos ellos entiendan.

El lenguaje que utilizaremos será el lenguaje definido por la FIPA: FIPA-SL [FIPA00]. Los protocolos utilizados serán comentados en la sección 7.7.1.

La ontología que utilizamos en el SMA Turist@ ha sido definida por nosotros mismos y la detallaremos en la sección 7.7.2.

7.7.1. Protocolos

El protocolo de comunicación define qué patrones determinados siguen los agentes que realizan el acto comunicativo. Es de utilidad para llevar el control de la conversación entre agentes de manera que cada agente sabe en todo momento que mensajes puede enviar y cuales puede recibir.

La FIPA como organismo normalizador ha definido una serie de protocolos [FIPAprotocols98] para su uso en la comunicación entre agentes. En nuestro caso tenemos dos situaciones bien diferenciadas, donde interviene la comunicación.

La primera consiste en el caso que un agente determinado pide a otro agente que realice una acción y que nos devuelva unos resultados. Del conjunto de protocolos definidos por la FIPA, el que más se ajusta a nuestras necesidades es el protocolo FIPA-Request.

La segunda situación se da cuando el Broker separa la tarea compleja en tareas sencillas y las envía a los agentes especializados, momento en el cual se espera a que todos los agentes especializados le contesten con los datos requeridos. Dicha situación podría resolverse con una serie de FIPA-Request, aunque de una forma excesivamente compleja para nuestros intereses. En vez de eso hemos optado por utilizar el FIPA-Contract Net, protocolo de negociación que nos dará de forma sencilla y eficiente, el comportamiento que queremos.

Estos protocolos fueron comentados en detalle en los apartados 3.6.1 y 3.6.3 donde hicimos una exposición sobre como JADE implementa dichos protocolos de comunicación.

7.7.2. Ontología

La ontología define el vocabulario u objetos utilizados, el conjunto de predicados que podemos formar y las acciones que se pueden solicitar a un agente.

En la ontología de nuestro SMA sólo necesitamos objetos y acciones. Nuestra ontología consistirá en la definición de los siguientes objetos o *frames*, formados por campos o *slots* de tipo simple, otros frames o un conjunto de elementos, que son detallados a continuación.

7.7.2.1. Objetos

Objeto Actividad

Proporciona información sobre parámetros de búsqueda.

Actividad		
Paràmetre	Tipus	Descripció
idcliente	string	Identificador del cliente que realiza la búsqueda
itinerario	boolean	Indicador de actividad presente
exhibicion	boolean	Indicador de actividad presente
museo	boolean	Indicador de actividad presente
monumento	boolean	Indicador de actividad presente
deporte	boolean	Indicador de actividad presente
teatro	boolean	Indicador de actividad presente
conferencia	boolean	Indicador de actividad presente
pelicula	boolean	Indicador de actividad presente
concierto	boolean	Indicador de actividad presente
precio	string	Requerimiento busqueda
fecha_ini	Date	Requerimiento busqueda
fecha_fi	Date	Requerimiento busqueda
id_pclaveMon	integer	Indicador temática busqueda
id_pclaveIti	integer	Indicador temática busqueda
id_pclaveMon	integer	Indicador temática busqueda
id_pclaveExh	integer	Indicador temática busqueda
id_pclaveMus	integer	Indicador temática busqueda
id_pclaveDep	integer	Indicador temática busqueda
id_pclaveTea	integer	Indicador temática busqueda
id_pclaveCon	integer	Indicador temática busqueda
id_pclavePel	integer	Indicador temática busqueda
id_pclaveCnc	integer	Indicador temática busqueda

Objeto Perfil

Proporciona información sobre el perfil de usuario.

Perfil		
Paràmetre	Tipus	Descripció
id_perfil	string	Identificador del perfil
fecha_ini	Date	Fecha inicio visita
fecha_fi	Date	Fecha final visita
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte
cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

Objeto Recomendación

Proporciona información para poder solicitar una recomendación.

Recomendación		
Paràmetre	Tipus	Descripció
idcliente	string	Identificador del usuario
coorx	integer	Coordenada posición
coory	integer	Coordenada posición
id_mapa	integer	Identificador zona

Objeto ListaItinerario

Proporciona una lista de objetos Itinerario.

ListaItinerario		
Paràmetre	Tipus	Descripció
lista_iti	Lista Itinerario	Lista de objetos itinerario

Objeto Itinerario

Proporciona información asociada a un Itinerario.

Itinerario		
Paràmetre	Tipus	Descripció
id	string	Identificador actividad
nombre	string	Nombre actividad
precio	string	Precio actividad
tipo	string	Temática de la actividad
coment	string	Comentarios
fecha_ini	string	Fecha inicio actividad
fecha_fi	string	Fecha final actividad
coorx	integer	Coordenada
coory	integer	Coordenada
id_mapa	integer	Identificador zona
zipcode	string	Código postal actividad
direc	string	Direccion actividad
ciudad	string	Ciudad donde se desarrolla la actividad
pais	string	País donde se desarrolla la actividad
descuento	string	Descuentos aplicables
lectivos	string	Horario
festivos	string	Horario
cerrado	string	Días cerrado
foto	string	Nombre fotografía asociada
nota	integer	Nota de la actividad
numvotos	integer	Número de votos
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte
cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

Objeto ListaMonumento

Proporciona una lista de objetos Monumento.

ListaMonumento		
Paràmetre	Tipus	Descripció
lista_mon	Lista Monumento	Lista de objetos monumento

Objeto Monumento

Proporciona información asociada a un Monumento.

Monumento		
Paràmetre	Tipus	Descripció
id	string	Identificador actividad
nombre	string	Nombre actividad
precio	string	Precio actividad
tipo	string	Temática de la actividad
coment	string	Comentarios
fecha_ini	string	Fecha inicio actividad
fecha_fi	string	Fecha final actividad
coorx	integer	Coordenada
coory	integer	Coordenada
id_mapa	integer	Identificador zona
zipcode	string	Código postal actividad
direc	string	Direccion actividad
ciudad	string	Ciudad donde se desarrolla la actividad
pais	string	País donde se desarrolla la actividad
descuento	string	Descuentos aplicables
lectivos	string	Horario
festivos	string	Horario
cerrado	string	Días cerrado
foto	string	Nombre fotografía asociada
nota	integer	Nota de la actividad
numvotos	integer	Número de votos
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte
cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

Objeto ListaMuseo

Proporciona una lista de objetos Museo.

ListaMuseo		
Paràmetre	Tipus	Descripció
lista_mus	Lista Museo	Lista de objetos museo

Objeto Museo

Proporciona información asociada a un Museo.

Museo		
Paràmetre	Tipus	Descripció
id	string	Identificador actividad
nombre	string	Nombre actividad
precio	string	Precio actividad

tipo	string	Temática de la actividad
coment	string	Comentarios
fecha_ini	string	Fecha inicio actividad
fecha_fi	string	Fecha final actividad
coorx	integer	Coordenada
coory	integer	Coordenada
id_mapa	integer	Identificador zona
zipcode	string	Código postal actividad
direc	string	Direccion actividad
ciudad	string	Ciudad donde se desarrolla la actividad
pais	string	País donde se desarrolla la actividad
descuento	string	Descuentos aplicables
lectivos	string	Horario
festivos	string	Horario
cerrado	string	Días cerrado
foto	string	Nombre fotografía asociada
nota	integer	Nota de la actividad
numvotos	integer	Número de votos
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte
cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

Objeto ListaExhibicion

Proporciona una lista de objetos Exhibición.

ListaExhibicion		
Paràmetre	Tipus	Descripció
lista_exh	Lista Exhibicion	Lista de objetos exhibición

Objeto Exhibicion

Proporciona información asociada a una Exhibición.

Exhibicion		
Paràmetre	Tipus	Descripció
id	string	Identificador actividad
nombre	string	Nombre actividad
precio	string	Precio actividad
tipo	string	Temática de la actividad
coment	string	Comentarios
fecha_ini	string	Fecha inicio actividad
fecha_fi	string	Fecha final actividad
coorx	integer	Coordenada
coory	integer	Coordenada
id_mapa	integer	Identificador zona
zipcode	string	Código postal actividad
direc	string	Direccion actividad
ciudad	string	Ciudad donde se desarrolla la actividad
pais	string	País donde se desarrolla la actividad

descuento	string	Descuentos aplicables
lectivos	string	Horario
festivos	string	Horario
cerrado	string	Días cerrado
foto	string	Nombre fotografía asociada
nota	integer	Nota de la actividad
numvotos	integer	Número de votos
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte
cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

Objeto ListaDeporte

Proporciona una lista de objetos Deporte.

ListaDeporte		
Paràmetre	Tipus	Descripció
lista_dep	Lista Deporte	Lista de objetos deporte

Objeto Deporte

Proporciona información asociada a un Deporte.

Deporte		
Paràmetre	Tipus	Descripció
id	String	Identificador actividad
nombre	String	Nombre actividad
precio	string	Precio actividad
tipo	string	Temática de la actividad
coment	string	Comentarios
fecha_ini	string	Fecha inicio actividad
fecha_fi	string	Fecha final actividad
coorx	integer	Coordenada
coory	integer	Coordenada
id_mapa	integer	Identificador zona
zipcode	string	Código postal actividad
direc	string	Direccion actividad
ciudad	string	Ciudad donde se desarrolla la actividad
pais	string	País donde se desarrolla la actividad
descuento	string	Descuentos aplicables
lectivos	string	Horario
festivos	string	Horario
cerrado	string	Días cerrado
foto	string	Nombre fotografía asociada
nota	integer	Nota de la actividad
numvotos	integer	Número de votos
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte

cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

Objeto ListaTeatro

Proporciona una lista de objetos Teatro.

ListaTeatro		
Paràmetre	Tipus	Descripció
lista_tea	Lista Teatro	Lista de objetos teatro

Objeto Teatro

Proporciona información asociada a una Teatro.

Teatro		
Paràmetre	Tipus	Descripció
id	string	Identificador actividad
nombre	string	Nombre actividad
precio	string	Precio actividad
tipo	string	Temática de la actividad
coment	string	Comentarios
fecha_ini	string	Fecha inicio actividad
fecha_fi	string	Fecha final actividad
coorx	integer	Coordenada
coory	integer	Coordenada
id_mapa	integer	Identificador zona
zipcode	string	Código postal actividad
direc	string	Direccion actividad
ciudad	string	Ciudad donde se desarrolla la actividad
pais	string	País donde se desarrolla la actividad
descuento	string	Descuentos aplicables
lectivos	string	Horario
festivos	string	Horario
cerrado	string	Días cerrado
foto	string	Nombre fotografía asociada
nota	integer	Nota de la actividad
numvotos	integer	Número de votos
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte
cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

Objeto ListaConferencia

Proporciona una lista de objetos Conferencia.

ListaConferencia		
Paràmetre	Tipus	Descripció
lista_con	Lista Conferencia	Lista de objetos conferencia

Objeto Conferencia

Proporciona información asociada a una Conferencia.

Conferencia		
Paràmetre	Tipus	Descripció
id	string	Identificador actividad
nombre	string	Nombre actividad
precio	string	Precio actividad
tipo	string	Temática de la actividad
coment	string	Comentarios
fecha_ini	string	Fecha inicio actividad
fecha_fi	string	Fecha final actividad
coorx	integer	Coordenada
coory	integer	Coordenada
id_mapa	integer	Identificador zona
zipcode	string	Código postal actividad
direc	string	Direccion actividad
ciudad	string	Ciudad donde se desarrolla la actividad
pais	string	País donde se desarrolla la actividad
descuento	string	Descuentos aplicables
lectivos	string	Horario
festivos	string	Horario
cerrado	string	Días cerrado
foto	string	Nombre fotografía asociada
nota	integer	Nota de la actividad
numvotos	integer	Número de votos
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte
cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

Objeto ListaPelicula

Proporciona una lista de objetos Película.

ListaPelicula		
Paràmetre	Tipus	Descripció
lista_pel	Lista Pelicula	Lista de objetos película

Objeto Pelicula

Proporciona información asociada a una Película.

Pelicula		
Paràmetre	Tipus	Descripció
id	string	Identificador actividad
nombre	string	Nombre actividad
precio	string	Precio actividad
tipo	string	Temática de la actividad
coment	string	Comentarios
fecha_ini	string	Fecha inicio actividad
fecha_fi	string	Fecha final actividad
coorx	integer	Coordenada
coory	integer	Coordenada
id_mapa	integer	Identificador zona
zipcode	string	Código postal actividad
direc	string	Direccion actividad
ciudad	string	Ciudad donde se desarrolla la actividad
pais	string	País donde se desarrolla la actividad
descuento	string	Descuentos aplicables
lectivos	string	Horario
festivos	string	Horario
cerrado	string	Días cerrado
foto	string	Nombre fotografía asociada
nota	integer	Nota de la actividad
numvotos	integer	Número de votos
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte
cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

Objeto ListaConcierto

Proporciona una lista de objetos Concierto.

ListaConcierto		
Paràmetre	Tipus	Descripció
lista_cnc	Lista Concierto	Lista de objetos concierto

Objeto Concierto

Proporciona información asociada a una Concierto.

Concierto		
Paràmetre	Tipus	Descripció
id	string	Identificador actividad
nombre	string	Nombre actividad
precio	string	Precio actividad
tipo	string	Temática de la actividad

coment	string	Comentarios
fecha_ini	string	Fecha inicio actividad
fecha_fi	string	Fecha final actividad
coorx	integer	Coordenada
coory	integer	Coordenada
id_mapa	integer	Identificador zona
zipcode	string	Código postal actividad
direc	string	Direccion actividad
ciudad	string	Ciudad donde se desarrolla la actividad
pais	string	País donde se desarrolla la actividad
descuento	string	Descuentos aplicables
lectivos	string	Horario
festivos	string	Horario
cerrado	string	Días cerrado
foto	string	Nombre fotografía asociada
nota	integer	Nota de la actividad
numvotos	integer	Número de votos
art	integer	Interés de la actividad en arte
history	integer	Interés de la actividad en historia
science	integer	Interés de la actividad en ciencia
music	integer	Interés de la actividad en música
sport	integer	Interés de la actividad en deporte
cinema	integer	Interés de la actividad en cine
theater	integer	Interés de la actividad en teatro

7.7.2.2. Acciones

Con los objetos definidos anteriormente no tenemos suficiente para poder establecer una comunicación eficiente entre los agentes. Necesitamos definir unas acciones para poder solicitar las funciones propias de cada agente.

Acción DevuelveActividades

Devuelve el paquete con las actividades para el agente personal.

DevuelveActividades		
Paràmetre	Tipus	Descripció
ListaItinerario	Lista Itinerario	Lista de objetos concierto
ListaMonumento	Lista Monumento	Lista de objetos monumento
ListaMuseo	Lista Museo	Lista de objetos museo
ListaExhibicion	Lista Exhibicion	Lista de objetos exhibición
ListaDeporte	Lista Deporte	Lista de objetos deporte
ListaTeatro	Lista Teatro	Lista de objetos teatro
ListaConferencia	Lista Conferencia	Lista de objetos conferencia
ListaPelicula	Lista Pelicula	Lista de objetos película
ListaConcierto	Lista Concierto	Lista de objetos concierto

Acción EnviaPerfil

Envía el perfil de un usuario.

EnviaPerfil		
Paràmetre	Tipus	Descripció
Perfil	Perfil	Perfil de usuario

Acción DevuelveIdPerfil

Devuelve el identificador de un usuario.

EnviaPerfil		
Paràmetre	Tipus	Descripció
id_perfil	string	Identificador de un perfil de usuario

Acción EnviaDesconexion

Informa de la desconexión de un usuario.

EnviaDesconexion		
Paràmetre	Tipus	Descripció
Perfil	Perfil	Perfil de usuario

Acción EnviaVotacion

Envía la valoración de una actividad.

EnviaDesconexion		
Paràmetre	Tipus	Descripció
idcliente	string	Identificador del usuario que hace la valoración
id_actividad	string	Identificador de la actividad valorada
categoria	string	Clase de actividad valorada
nota	integer	Nota dada a la actividad

Acción BuscaActividad

Realiza una consulta de actividades.

BuscaActividad		
Paràmetre	Tipus	Descripció
Actividad	Actividad	Objeto con las actividades a buscar

Acción BuscaRecomendacion

Solicita una recomendación.

BuscaRecomendacion		
Paràmetre	Tipus	Descripció
Recomendacion	Recomendacion	Objeto con los parámetros de la recomendación

Acción DevuelveListaItinerario

Envía una lista de itinerarios.

DevuelveListaitinerario		
Paràmetre	Tipus	Descripció
ListaItinerario	ListaItinerario	Lista itinerarios

Acción DevuelveListaMonumento

Envía una lista de monumento.

DevuelveListaMonumento		
Paràmetre	Tipus	Descripció
ListaMonumento	ListaMonumento	Lista monumentos

Acción DevuelveListaMuseo

Envía una lista de museos.

DevuelveListaMuseo		
Paràmetre	Tipus	Descripció
ListaMuseo	ListaMuseo	Lista museos

Acción DevuelveListaExhibicion

Envía una lista de exhibición.

DevuelveListaExhibicion		
Paràmetre	Tipus	Descripció
ListaExhibicion	ListaExhibicion	Lista exhibición

Acción DevuelveListaDeporte

Envía una lista de deporte.

DevuelveListaDeporte		
Paràmetre	Tipus	Descripció
ListaDeporte	ListaDeporte	Lista deportes

Acción DevuelveListaTeatro

Envía una lista de teatro.

DevuelveListaTeatro		
Paràmetre	Tipus	Descripció
ListaTeatro	ListaTeatro	Lista teatros

Acción DevuelveListaConferencia

Envía una lista de conferencia.

DevuelveListaConferencia		
Paràmetre	Tipus	Descripció
ListaConferencia	ListaConferencia	Lista conferencias

Acción DevuelveListaPelicula

Envía una lista de películas.

DevuelveListaPelicula		
Paràmetre	Tipus	Descripció
ListaPelicula	ListaPelicula	Lista películas

Acción DevuelveListaConcierto

Envía una lista de conciertos.

DevuelveListaConcierto		
Paràmetre	Tipus	Descripció
ListaConcierto	ListaConcierto	Lista conciertos

Acción Respuesta

Envía un mensaje cualquiera.

Respuesta		
Paràmetre	Tipus	Descripció
mensa	string	Mensaje cualquiera

8. EL SISTEMA DE RECOMENDACIÓN

Nuestro sistema de recomendación se centra en turistas que visitan esporádicamente la ciudad; por lo tanto, no tenemos necesidad de almacenar ficheros históricos o de un largo periodo de tiempo.

8.1. El perfil de usuario

El perfil asociado a un usuario está compuesto por la información más importante referente a su estancia vacacional. Dichos datos deben ser los mínimos posibles dado que inicialmente son entrados por el propio turista, al arrancar por primera vez la aplicación, y debido a la naturaleza de los dispositivos móviles, no podemos sobrecargar la pantalla con detalles superfluos. En este caso los datos que pedimos son:

- Fecha inicio del viaje
- Fecha finalización del viaje
- Interés del usuario en arte
- Interés del usuario en historia
- Interés del usuario en ciencia
- Interés del usuario en música
- Interés del usuario en deportes
- Interés del usuario en cine
- Interés del usuario en teatro

Los campos referentes a interés se almacenan como valores entre 0 y 10, siendo 0 el interés más bajo y 10 el más elevado. Dichos intereses se tratan internamente como un vector de enteros, lo cual nos permitiría escalar el número de campos de interés en caso de necesidad.

Cada actividad turística contará, a su vez, con un vector de enteros con los mismos campos que el vector asociado al perfil de usuario posee y que ya hemos nombrado. La comparación efectuada entre el vector con los intereses del usuario y el vector con los intereses asociados a la actividad será la base del proceso de recomendación.

8.2. Generación del perfil inicial

Cuando se incorpora un nuevo usuario al sistema, éste almacenará la información asociada a sus preferencias y características para procesar la generación del perfil del turista. El sistema preguntará al usuario los campos necesarios, mediante un formulario que el asistente personal, ejecutándose en el dispositivo móvil, mostrará y que el turista rellenará antes de poder empezar a utilizar la aplicación.

La generación del perfil sólo se produce la primera vez que el turista accede al sistema. El perfil queda almacenado en el móvil utilizando técnicas de persistencia de datos, y de igual forma el agente recomendador guarda una copia del mismo, llevando el control de si el turista está conectado al sistema o no (no informamos de actividades a usuarios desconectados).

El asistente personal permite al turista modificar en cualquier momento su perfil, lo cual actualizará automáticamente la copia situada en la base de datos local del recomendador consiguiendo así coherencia en la gestión de los perfiles.

El perfil del turista va variando con el tiempo, adaptándose a los gustos del usuario, a medida que

éste realiza consultas y valora las diversas actividades que se le van mostrando. De esta manera obtenemos un sistema de recomendación adaptativo, que aprende y mejora sus prestaciones con el tiempo.

8.3. Actualización del perfil por aprendizaje

Como ya hemos dicho el usuario en cualquier momento puede consultar su perfil actual y modificarlo como crea conveniente. No obstante el sistema utiliza aprendizaje para actualizar dicho perfil automáticamente. Esto se consigue gracias a que el asistente personal tiene en cuenta las acciones realizadas por el turista y las transmite al agente recomendador cuando lo cree necesario. El recomendador gestionará dichas acciones y decidirá si el perfil debe ser modificado o no, informando al asistente personal del nuevo perfil disponible, en caso afirmativo.

Los dos métodos que el sistema utiliza para recoger la información necesaria para modificar un perfil de usuario son los siguientes:

- Explícitamente: El usuario hace una valoración de una determinada actividad mostrando explícitamente cuales son sus preferencias.
- Implícitamente: El asistente personal analiza las consultas que realiza el usuario y extrae de ellas sus preferencias.

8.3.1. Actualización explícita del perfil

El asistente personal ofrece una agenda al turista en la cual éste puede introducir las actividades que son de su interés. Desde dicha agenda, el usuario puede consultar en cualquier momento la información asociada a la actividad presente, de esta manera buscamos agilizar el tratamiento de las actividades que el turista esta interesado en realizar, evitando que deba realizar una búsqueda adicional cada vez que quiera saber algo de dicha actividad.

No obstante el trabajo de la agenda no finaliza aquí. Desde ella, el turista podrá valorar las actividades presentes, que son las que consideramos sus favoritas. La valoración consiste en dar una nota del 0 (no le gusta) al 10 (le gusta mucho) en referencia a la actividad en cuestión. Dicha valoración llegará tanto al recomendador que actualizara su base de datos con la nota, como al agente encargado de dicha actividad, que también procederá a actualizar su base de datos local. Con esto conseguimos mantener la coherencia de los datos, manteniendo siempre el perfil de la actividad actualizado en ambas fuentes.

La nota que le damos a una actividad queda por tanto computada en su perfil. Dicha nota será mostrada al consultar la información asociada a la actividad en cuestión y de esta forma sabremos la valoración global que ha tenido por parte de los demás turistas, dándonos un punto extra de información a la hora de elegir qué ver o qué visitar.

No obstante, la nota dada también le sirve al sistema para actualizar el perfil del usuario. Para ello el sistema tendrá en cuenta el vector de intereses asociado a la actividad valorada y la nota que haya dado el turista. La manera de computar esto es la siguiente. Para cada atributo de la actividad (arte, historia, ciencia, música, deportes, cine, teatro) se mira si supera un cierto limite por el cual lo consideramos un atributo fuerte: por ejemplo consideramos que un atributo por encima de 5 (la escala es de 0 a 10) se considera fuerte en la actividad y será candidato a modificar el perfil de usuario en ese mismo atributo, según la valoración que el turista haya realizado: notas muy negativas harán descender el atributo del perfil de usuario, mientras que notas muy favorables lo aumentarán.

Veamos un ejemplo del funcionamiento para dejar claro como se actualiza explícitamente el perfil de usuario:

Suponemos que tenemos un vector u con los grados de interés del usuario en los diferentes aspectos culturales y de ocio comentados, donde u_i es un valor entre 0 y 10 que representa el interés en el aspecto i y un vector a con las características de una actividad, donde a_i es la valoración de la característica de la actividad.

Consideremos el atributo i-esimo que tiene asociado un cierto valor que indica su importancia, con lo que tendríamos:

Si $a_i > 5 \rightarrow u_i = u_i + \theta$

Donde θ variara en función de la valoración del usuario según sea una nota pésima (<2) que restará un punto, o sea una nota muy favorable (>8) en cuyo caso sumará un punto.

Por ejemplo, tenemos una actividad con un interés en arte de 7 y el interés del usuario en arte es 4. Suponemos que el turista ha valorado la actividad con un 1 (realmente no le ha gustado nada). Como el interés de la actividad es mayor que 5 (el arte es atributo fuerte en esta actividad) el interés del usuario en arte disminuirá 1 punto, cambio que queda reflejado en su perfil actual.

Evidentemente, como al usuario no le ha gustado una actividad bien valorada en arte, se le disminuye en su perfil el interés en arte.

8.3.2. Actualización implícita del perfil

En este caso la actualización del perfil se realiza a partir de la observación que realiza el sistema de las acciones que lleva a cabo el usuario.

El recomendador recibe la notificación de las búsquedas que realiza el turista y busca en ellas temáticas clave que el considere importantes a la hora de modificar el perfil asociado al usuario.

Los tipos de actividad y la temática asociada sobre la cual se activará la actualización implícita del perfil son los siguientes:

- *Monumentos:* búsqueda de monumentos relacionados con eventos históricos.
- *Itinerarios:* búsqueda de itinerarios que se realizan por lugares de interés histórico.
- *Museos:* búsqueda de museos relacionados con el arte, la historia o la ciencia.
- *Exhibiciones:* búsqueda de exposiciones y actividades relacionadas con la historia.
- *Teatro:* búsqueda de obras de teatro con un fuerte componente musical.
- *Conferencia:* búsqueda de conferencias relacionadas con el arte, la historia o la ciencia.

La actualización del perfil se llevará acabo cuando el turista indique en sus criterios de búsqueda, la temática específica asociada a cada uno de los tipos.

Por ejemplo, un usuario que busque monumentos, especificando que solo quiere ser informado de aquellos cuya temática sea la historia, merecerá el aumento en un punto de su interés en historia, por considerar que realmente esta muy interesado en este tema en concreto.

8.4. Cuando se recomienda

El proceso de recomendación se dispara en diversas circunstancias que pasamos a comentar brevemente:

1. Nueva actividad: Al introducirse una nueva actividad en el sistema el recomendador analiza si dicha actividad es del interés de alguno de los usuarios conectados, en caso afirmativo notifica al turista de la nueva actividad disponible.

2. Recomendación explicita: El turista pide a su asistente personal que le recomiende una serie de actividades que se ajusten a su perfil.

3. Localización del turista en la ciudad: Cuando el turista pregunta al asistente personal su localización actual en la ciudad. El asistente obtiene la zona donde se encuentra y pregunta al recomendador por actividades que sean del interés del usuario y que se encuentren en dicha zona. Una vez el asistente tiene toda la información, muestra el mapa de la zona en cuestión resaltando sobre dicho mapa la posición de las actividades recomendadas.

4. Cambio de zona por parte del turista: El asistente personal, mediante su sistema localizador, tendrá en cuenta cuando el turista, en su trayecto por la ciudad, cambia de zona, y realizará una consulta al recomendador en busca de actividades presentes en la nueva zona que puedan ser del interés del usuario, avisándole en caso afirmativo.

8.5. Cómo se recomienda

En este punto comentaremos el proceso que sigue el recomendador para decidir cuando una actividad se adapta al perfil de un usuario y por lo tanto, puede ser recomendada.

En nuestro sistema las recomendaciones se basarán en los resultados obtenidos de la comparación de los vectores de características de las actividades con los vectores de intereses de los usuarios. Para ello utilizaremos una distancia Euclidiana de los atributos de los vectores, teniendo en cuenta que los atributos de una actividad iguales a 0, no serán tenidos en cuenta en el cómputo total de la distancia. De esta manera buscamos darle importancia a los atributos que realmente aporten algo a la actividad, mejorando enormemente el rendimiento del proceso de recomendación.

Para explicarlo más en detalle, supongamos que tenemos un vector u con los grados de interés del usuario en diferentes aspectos culturales y de ocio, donde u_i es un valor entre 0 y 10 que representa el interés en el aspecto i y un vector a con las características de una actividad, donde a_i es la valoración de la característica i de la actividad.

Podemos calcular la similitud entre los gustos de un usuario y la descripción de una actividad realizando el calculo de la distancia euclidiana entre estos dos vectores y dividiendo por la raíz de la máxima distancia entre los dos vectores, es decir, el número de atributos de los vectores que tenemos en cuenta (atributos de actividad iguales a 0 no se tienen en cuenta en el cálculo) y que llamaremos n.

$$\text{Distancia } \delta = \frac{\sqrt{\sum_{i=1}^{n} (u_i - a_i)^2}}{\sqrt{n}}$$

Una vez obtenida la distancia entre los dos vectores, tendremos un valor decimal entre 0 y 10.0, que nos indicará lo cercanos que están los dos vectores entre sí. Cuanto más pequeño sea este valor más cercanos estarán ambos vectores entre sí, o lo que es lo mismo, más cerca estarán los gustos del usuario con las características ofrecidas por la actividad. Nuestra tarea ahora será determinar a partir de que distancia recomendamos una actividad. Por ejemplo, podemos decidir recomendar aquellas actividades que obtengan una distancia menor a 4.8 respecto al vector de intereses del usuario; descartando todas las que obtengan un valor por encima de esa cifra.

9. EL SISTEMA DE LOCALIZACIÓN

Una de las funcionalidades más destacadas de nuestro aplicativo es la capacidad de conocer la posición aproximada del turista en la ciudad, mostrando un mapa (imagen en formato png) del área donde éste se encuentra y resaltando actividades en dicha zona que sean consideradas de interés para el turista.

La primera idea que nos viene a la cabeza para conseguir esto, es utilizar un módulo GPS[20] [GPS05] que mediante el uso de satélites nos de la posición actual del usuario. El GPS es sin duda la opción que más precisión nos daría, estaríamos hablando de entre 4 y 40 metros de error. No obstante el GPS presenta serios inconvenientes: Estamos hablando de un dispositivo que necesita tener visibles los satélites por lo tanto en ciudad sufriremos el llamado "efecto cañón" dado que la presencia de edificios anulará la visual entre el módulo GPS y los satélites. Siguiendo en esta línea, el sistema no podría funcionar dentro de un lugar cerrado, incluso siendo este lugar el bolsillo del pantalón. Para solventar esto, dependeríamos de una mejora del dispositivo, que requeriría un desembolso adicional además de aumentar la complejidad de la supuesta solución. Otros inconvenientes del uso de GPS es el hecho que arrancando en frío, requiere un tiempo para triangularse con los satélites, es además un elemento extra a llevar con nosotros y si por contra, se encuentra integrado en el dispositivo móvil, realizará un consumo extra de la batería, sin olvidar que estamos hablando de un elemento de hardware adicional, que nos constará un dinero obtenerlo y un tiempo extra configurarlo.

La alternativa que nosotros hemos integrado en nuestro aplicativo, es la utilización de la propia red GSM [GSM05] que utiliza el móvil para realizar y recibir llamadas. Esto nos simplifica la solución, puesto que no tenemos que añadir nada más a nuestro móvil, simplemente utilizamos una de sus funcionalidades en nuestro beneficio. Por lo tanto no requerirá gastar dinero extra, nos aprovecharemos de lo que ya tenemos. El inconveniente de este método es su limitada precisión, el GSM no se basa en una alta resolución, sino que está enfocado para trabajar con áreas que tendrán un tamaño de cientos de metros. Esto que a priori puede parecer poco útil habría que trasladarlo a lo que queremos conseguir: Tenemos un turista caminando por una ciudad y lo que queremos es mostrarle su posición y las actividades que se encuentran en la zona por la que se desplaza. El concepto de zona se adapta perfectamente a lo que es una celda GSM. Podemos asociar cada celda con una zona, y mostrar de esta manera al turista un mapa de dicha zona con las actividades presentes en esa zona. Visto así, vemos que la red GSM sí se adapta perfectamente a nuestras necesidades, ofreciéndonos además grandes ventajas y es que el sistema de localización funcionará en cualquier lugar donde haya cobertura GSM, sea dentro de un edificio o en cualquier calle estrecha, desmarcándonos así de los inconvenientes del GPS. Y recordamos que no requiere ningún desembolso extra, puesto que la tecnología GSM esta integrada en todos los móviles actuales.

Entremos ahora a describir más en detalle el sistema. La cobertura GSM está particionada en celdas que cubren toda la ciudad. El móvil, para obtener dicha cobertura, se conecta a la celda más potente que encuentra en la zona y sabemos que las celdas tienen un identificador único y siempre están situadas en el mismo lugar. El móvil siempre sabe en que celda esta conectado, por lo tanto la tarea de nuestro sistema de localización será obtener del dispositivo móvil, el identificador de la celda donde nos encontramos, asociándola a la zona de la ciudad donde estamos.

¿Como obtenemos el identificador del móvil? Nuestros agentes están implementados con Java, lo inteligente sería encontrar una librería que trabajase con el módulo GSM del móvil. Hay diversos artículos en la red, que discuten las posibles alternativas. Destacamos el J2ME and Location-Based Services de Qusay H. Mahmoud [Mahmoud05] publicado en la Sun Developers Network [SunDev05]. De aquí podemos extraer la existencia de la Location API for J2ME (JSR 179) que podemos consultar

[20] *Global Positioning System.*

en la página web de la Java Comunity Process: http://www.jcp.org/en/jsr/detail?id=179. Dicha API aun no esta incluida como paquete opcional dentro del entorno de desarrollo para J2ME de Sun: el J2ME Wireless Toolkit. No obstante podemos bajar el Jar asociado a dicha API en la página web del Forum Nokia: http://www.forum.nokia.com/main/1,,1_0_10,00.html#jsr179.

De todas maneras, pese a la existencia de dicha API, de la lectura del artículo de Mahmoud nos percatamos de un severo contratiempo: La API necesita CLDC-1.1 para el uso de coma flotante en las coordenadas de posición, como podemos ver en el texto extraído del mismo artículo:

> *JSR 179 requires the Connected Device Configuration (CDC) or version 1.1 of the Connected Limited Device Configuration (CLDC). CLDC 1.0 isn't adequate because it doesn't support floating-point numbers, which the API uses to represent coordinates and other measurements.*

El dispositivo móvil con que trabajamos, la Nokia N-GAGE, solo soporta CLDC-1.0 como se comenta en el capitulo dedicado a nuestro dispositivo. Por lo tanto dicha API no nos sirve y tendremos que utilizar ideas alternativas.

Aparcado Java para obtener el identificador, pasamos a analizar lo que el entorno de programación Symbian nos ofrece. En este punto es de vital importancia la aportación de Mika Raento [Raento05] y sus explicaciones sobre C++ bajo el sistema operativo Symbian. Gracias a sus ejemplos y tutoriales encontramos la manera de obtener el identificador de celda GSM mediante las librerías *gsmbas.lib* y *etel.lib* presentes en el entorno Symbian de nuestro dispositivo móvil. Dichas librerías nos dan el identificador de celda y el código de área asociado a dicha celda (código que corresponde con la ciudad en cuestión).

Por lo tanto implementaremos el sistema de localización en C++ con llamadas a las librerías aportadas por Symbian, y enviaremos los identificadores de celda obtenidos al asistente personal, implementado en Java, mediante el uso de sockets dentro del propio dispositivo.

Cabe destacar, al hablar de la conectividad vía sockets, la problemática asociada a las diferencias en los formatos de datos en C++ y Java. Esta problemática con la que topamos debe ser resuelta para poder comunicar por socket ambos entornos; en los siguientes párrafos pasamos a comentar las soluciones aplicadas en este caso.

Lo normal es que un entero, por ejemplo, sean 4 bytes aunque en micros antiguos eran de 2 bytes y los más modernos empiezan a ser de 8 bytes. Si mandamos un entero a través de un socket, estamos enviando estos 4 bytes. Si los micros de ambos lados del socket tienen el mismo orden para los bytes no hay problema, pero si tienen distinto, el entero se interpretará incorrectamente.

La máquina virtual Java, por ejemplo, define los char como 2 bytes (para poder utilizar caracteres UNICODE), mientras que en el resto de micros habituales suele ser de un byte. Si enviamos desde Java un carácter a través de un socket, enviamos 2 bytes, mientras que si lo leemos del socket desde un programa en C, solo leeremos un byte, dejando el otro pendiente de lectura, lo cual producirá un claro efecto adverso en nuestro sistema.

En el caso de floats y doubles la cosa se complica mucho más. El formato aplicado en estos casos puede variar mucho de un entorno a otro y la conversión se convierte en algo nada trivial. En general podemos concluir que, suele ser buena idea, si queremos comunicar máquinas distintas, hacer que no se envíen floats o doubles.

La solución para evitar todo esto, será enviar los datos de una forma estándar e independiente del entorno en que trabajemos. Para ello en C tenemos las funciones:

- *htonl()* pasa un entero de formato hardware a formato red.
- *ntohl()* pasa un entero de formato red a formato hardware.
- *htons()* hace lo mismo que htonl() pero con un short (2 bytes).
- *ntohs()* hace lo mismo que ntohl() pero con un short (2 bytes).

En Java, a partir de la clase DataInputStream, que sí tenemos en MIDP-1.0 con CLDC-1.0, podremos obtener la información que nos envíe el módulo implementado en C++.

Llegados a este punto, el asistente personal ya es capaz de recibir los identificadores que envía el módulo en C++. Su trabajo ahora es, mediante un thread, reconocer la celda actual e informar de la zona de la ciudad a la cual corresponde, que de hecho, será lo que finalmente se mostrará al turista. Para ello, el asistente personal, obtiene de un fichero de texto, la asociación entre identificadores y zonas. El fichero también incluye el identificador del mapa asociado a cada zona, de esta manera el asistente sabe en todo momento que mapa debe mostrar al turista.

Por lo tanto, para añadir nuevas zonas al asistente personal, solamente tendremos que añadir la entrada al fichero de texto, con el identificador de la celda, su traducción a zona y el mapa al cual queda asociado. El funcionamiento interno es sencillo, al arrancar el asistente personal lee el fichero de texto e introduce todos los datos en una tabla de hash, así conseguimos que cuando el módulo en C++ envíe el código de la celda actual, el asistente obtenga la zona y el mapa asociado rápida y eficientemente.

Como último apunte sobre el sistema localizador, comentar que aunque el formato actual con un módulo en C++ obteniendo el identificador de celda funciona correctamente, sólo es inteligente aplicarlo en dispositivos móviles limitados en cuanto al uso de Java se refiere, como es nuestro caso, que trabajamos con MIDP-1.0 y CLDC-1.0. Para móviles más modernos que estén dotados de MIDP-2.0 y CLDC-1.1 debe ser el mismo agente personal quien, utilizando la Location API for J2ME, obtenga directamente el identificador de celda GSM, de esta forma tendríamos un sistema más robusto y eficiente.

10. DEMOSTRACIÓN PRÁCTICA

10.1. Inicialización del sistema

Al iniciar el sistema en el servidor central, todos los agentes que intervienen en el SMA, exceptuando el del asistente personal que debe ser arrancado dentro del dispositivo móvil, se registran en el DF y arrancan sus bases de datos locales, preparándose para ofrecer los servicios que cada uno tiene asignado. Una vez en funcionamiento todos los agentes de actividades envían sus actividades al agente recomendador para que éste tenga la información actualizada y coherente.

Mostramos la salida por consola de este proceso donde se ve claramente diferenciados el proceso de registrar los agentes y el de informar al recomendador:

```
Agent container Main-Container@JADE-IMTP://AlexMobile is ready.
AGENTE DE PELICULAS REGISTRADO EN EL DF
AGENTE DE TEATROS REGISTRADO EN EL DF
AGENTE DE EXHIBICIONES REGISTRADO EN EL DF
AGENTE DE MUSEOS REGISTRADO EN EL DF
AGENTE DE CONCIERTOS REGISTRADO EN EL DF
BROKER REGISTRADO EN EL DF
AGENTE DE CONFERENCIAS REGISTRADO EN EL DF
RECOMMENDER REGISTRADO EN EL DF
AGENTE DE DEPORTES REGISTRADO EN EL DF
AGENTE DE ITINERARIOS REGISTRADO EN EL DF
AGENTE DE MONUMENTOS REGISTRADO EN EL DF
Recommender:Nuevo/s Conferencia/s Guardado/s
Recommender:Nuevo/s Exhibicion/s Guardado/s
Recommender:Nuevo/s Teatro/s Guardado/s
Recommender:Nuevo/s Monumento/s Guardado/s
Recommender:Nuevo/s Deporte/s Guardado/s
Recommender:Nuevo/s Concierto/s Guardado/s
Recommender:Nuevo/s Pelicula/s Guardado/s
Recommender:Nuevo/s Itinerario/s Guardado/s
Recommender:Nuevo/s Museo/s Guardado/s
```

10.2. Creación perfil de usuario

Al arrancar el asistente personal por primera vez, el sistema nos mostrará un formulario donde introducir los datos asociados al nuestro perfil: Fechas de la estancia en la ciudad (por defecto la fecha de inicio del viaje será el día actual y la fecha de finalización un mes después) y valoración de nuestros intereses. Para esta prueba valoraremos con un 5 nuestro interés en arte y ciencia, mientras que le daremos un 2 a nuestro interés en historia. Los demás parámetros (música, deporte, cine y teatro) los dejaremos a 0.

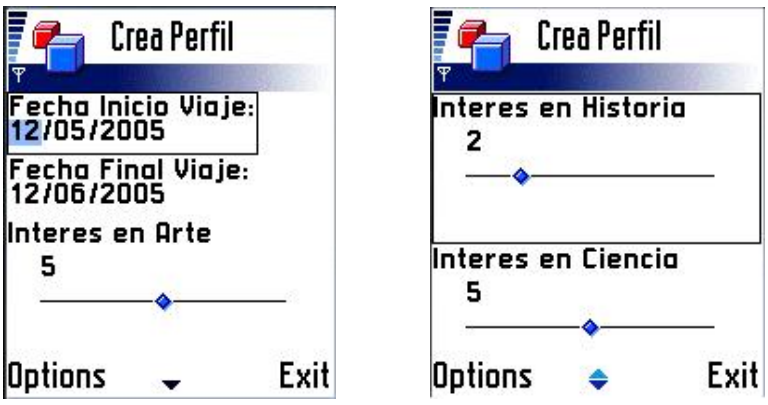


Fig. 22 Formulario para la creación del perfil

Una vez guardamos el perfil, éste se envía al recomendador y nos conectaremos al sistema, mostrándose el mensaje en la consola del servidor:

```
Cliente1115910107548 con perfil: id1413 ONLINE!!
```

Donde podemos ver el identificador de perfil que nos ha sido asignado y que conservaremos hasta que desinstalemos el asistente personal.

10.3. Pantalla y menú principal

Una vez accedemos al sistema podemos contemplar la pantalla inicial y podremos acceder al menú principal del asistente personal desde donde podremos:

- Conocer nuestra posición actual.
- Buscar actividades según unos criterios de busqueda.
- Solicitar una recomendación acorde con nuestro perfil de usuario.
- Mirar avisos del sistema: actividades de interés en la zona o nuevas actividades introducidas.
- Ver nuestras actividades favoritas en la agenda.
- Modificar nuestro perfil.
- Visualizar el About del sistema.
- Salir del sistema.

Fig. 23 Pantalla y menú principal

10.4. Recomendación

Desde el menú principal solicitaremos al sistema que nos presente una serie de actividades recomendadas que se ajusten a nuestro perfil. El agente recomendador analizará las actividades del sistema y comparará sus parámetros con los del usuario que solicita la recomendación. En la salida de consola del servidor podemos ver el proceso de recomendación:

```
Recommender: id1413 quiere recomendacion
Recommender: RECOMIENDA Amphitheater distancia: 3.366501646120693
Recommender: NO RECOMIENDA Aqueduct distancia: 6.164414002968977
Recommender: NO RECOMIENDA Cathedral distancia: 5.354126134736338
Recommender: RECOMIENDA Sescelades distancia: 4.43471156521669
Recommender: RECOMIENDA City1 distancia: 3.8729833462074175
Recommender: RECOMIENDA Playas distancia: 3.7080992435478315
```

```
Recommender: RECOMIENDA Dones a Tarraco distancia: 3.559026084010437
Recommender: RECOMIENDA Museu de Historia distancia: 2.9325756597230357
Recommender: NO RECOMIENDA Nastic-Recreativo distancia: 5.809475019311125
Recommender: NO RECOMIENDA La ferida lluminosa distancia: 5.431390245600108
Recommender: RECOMIENDA Farmacia i pacients distancia: 3.3166247903554
Recommender: NO RECOMIENDA Ninotchka distancia: 5.431390245600108
Recommender: NO RECOMIENDA Orquesta simfonica distancia: 5.361902647381804
Recommender devuelve recomendacion
```

Recordemos que los intereses del usuario son 5 para arte, ciencia y 2 para historia. Hemos dado la orden al agente recomendador que sólo recomiende las actividades con una distancia por debajo de 4.8. La lista de actividades recomendadas se muestra al usuario en la pantalla correspondiente a la figura 24.

Fig. 24 Lista actividades recomendadas

Desde aquí el turista puede seleccionar la actividad que desee, ver sus detalles y acceder al menú de actividad, como se muestra en la figura 25:

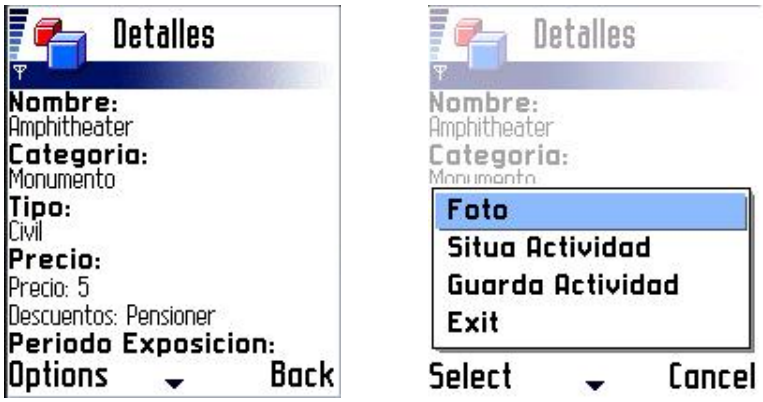


Fig. 25 Detalles y menú de actividad

Desde el menú de actividad podemos ver la foto asociada a la actividad, en este caso el anfiteatro de Tarragona; Situar la actividad en el mapa de la ciudad o guardar la actividad en nuestra agenda. Realizaremos las tres acciones cuyos resultados mostramos en la figura 26. Podemos destacar que una vez dentro de la pantalla de localización, con los cursores del móvil, nos podemos desplazar por el

mapa asociado al área donde se sitúa la actividad, de esta manera facilitamos la orientación del turista.

Fig. 25 Foto, localización y aviso de actividad guardada en agenda

10.5. Búsqueda de actividades

Desde el menú principal elegiremos la opción “Buscar actividades”, nos saldrá un formulario con los criterios de búsqueda, como podemos ver en la figura 26. Seleccionaremos ver exhibiciones y elegiremos explícitamente la temática historia. Hay que destacar que cada tipo de actividad tiene su temática asociada y para aprovechar al máximo las dimensiones de la pantalla, la selección de tema, solo aparece una vez seleccionamos los tipos de actividad que queremos buscar.

Fig. 26 Formulario para la búsqueda de actividades

Al realizar la búsqueda el recomender será informado de nuestra selección y tras comprobar que estamos interesados en historia, aumentará su valoración en nuestro perfil, lo cual queda reflejado por la consola del servidor:

```
Recommender: Aumentamos el parametro <Historia> para id1413
```

Cambio que puede verse reflejado desde el menú “Modifica Perfil” desde donde podemos editar o simplemente visualizar nuestro perfil de usuario y que ahora refleja un interés en historia de 3 puntos, como podemos ver en la figura 27.

Fig. 27 Perfil actual

Tras la búsqueda realizada, el asistente nos muestra una lista con las actividades que se ajustan a nuestros criterios. En este caso, encontramos una exhibición de la cual disponemos las mismas opciones ya mostradas en el punto *10.4 Recomendación*. Podemos ver dicho resultado en la figura 28.

Fig. 26 Resultado de la búsqueda

Si ahora volvemos a solicitar una recomendación, el sistema tendrá en cuenta el aumento experimentado del interés en historia del usuario. Obteniendo el siguiente resultado por la consola del servidor:

```
Recommender: id1413 quiere recomendacion
Recommender: RECOMIENDA Amphitheater distancia: 2.943920288775949
Recommender: NO RECOMIENDA Aqueduct distancia: 5.354126134736338
Recommender: RECOMIENDA Cathedral distancia: 4.69041575982343
Recommender: RECOMIENDA Sescelades distancia: 4.1231056256176615
Recommender: RECOMIENDA City1 distancia: 3.3166247903554
Recommender: RECOMIENDA Playas distancia: 4.092676385936225
Recommender: RECOMIENDA Dones a Tarraco distancia: 4.08248290463863
Recommender: RECOMIENDA Museu de Historia distancia: 2.4899799195977463
Recommender: NO RECOMIENDA Nastic-Recreativo distancia: 6.06217782649107
Recommender: NO RECOMIENDA La ferida lluminosa distancia: 5.70087712549569
Recommender: RECOMIENDA Farmacia i pacients distancia: 3.8729833462074175
Recommender: NO RECOMIENDA Ninotchka distancia: 5.70087712549569
Recommender: NO RECOMIENDA Orquesta simfonica distancia: 5.545268253204709
Recommender devuelve recomendacion
```

Gracias al aumento en el campo historia, el sistema recomendador considera que la Catedral ahora sí puede ser de nuestro interés y por tanto nos la adjunta con el resto de recomendaciones, que podemos consultar siguiendo el procedimiento ya comentado en el punto *10.4 Recomendación*.

10.6. Localización

Desde el menú principal seleccionamos la opción “Donde Estoy”, con lo que el asistente personal consulta la posición actual del turista y pregunta al recomendador que actividades hay en la zona que se adapten a su perfil. El resultado queda mostrado en la figura 27, donde vemos dos actividades de interés en la zona del campus Sescelades (URV). Desde esta pantalla podemos tanto ver los detalles de las actividades usando el procedimiento habitual, como ver el mapa de la zona donde nos encontramos con las actividades recomendadas resaltadas.

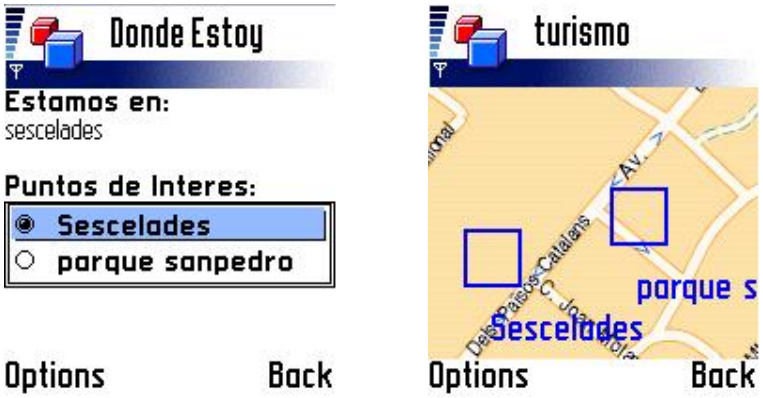


Fig. 26 Localización y mapa

Como es habitual, con los cursores del móvil podremos desplazarnos por el mapa que indica la zona en donde estamos aproximadamente situados.

10.7. Avisos del sistema

El recomendador avisará al turista automáticamente cuando se den dos circunstancias especiales que se consideren de interés para el usuario:

- Se introduce en el sistema una nueva actividad que se adapta al perfil del usuario.
- El usuario entra en una zona con actividades que se adaptan a su perfil.

Cuando se den estas circunstancias, el móvil lanzará una señal acústica y una pantalla de dialog avisará del nuevo evento. Las actividades que así nos llegan se pueden ver en la opción “Avisos” del menú principal. Podemos ver que al lado de “Avisos” hay un símbolo de admiración que indica que el turista ha recibido avisos que aun no han sido vistos. Una vez el usuario vea los avisos disponibles el símbolo de admiración desaparecerá hasta recibir nuevos eventos.

Eventos que ya se encuentren pendientes de visualizar, no se volverán a contabilizar. Una vez dentro de la pantalla de Avisos, se nos mostrarán las actividades en una lista con el formato habitual y podremos consultar sus detalles asociados como hemos hecho hasta ahora. En la figura 27 podemos ver 3 pantallas que exponen lo comentado.

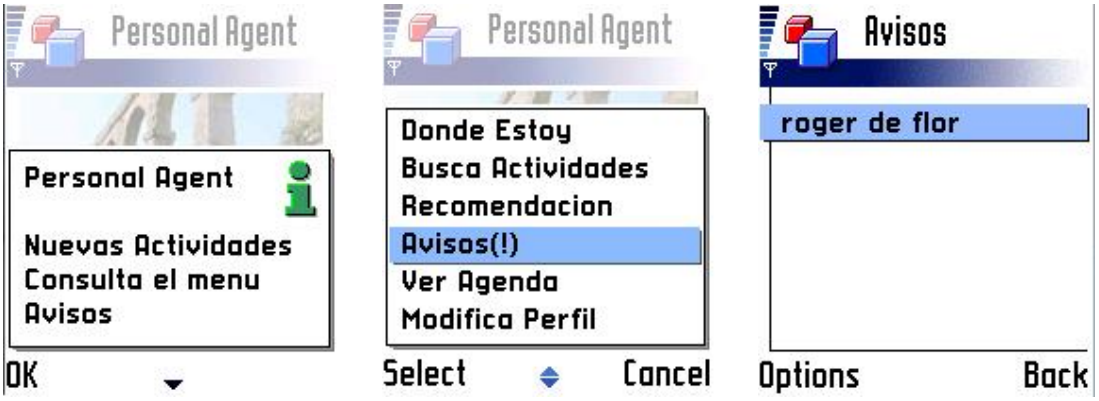


Fig. 27 Avisos

10.8. Agenda y votaciones

Desde el menú principal seleccionaremos la opción “Agenda”, esto nos mostrará la lista de actividades que el usuario ha ido guardando por considerarlas favoritas. Estas actividades se guardan en memoria del móvil y por tanto el turista puede consultarlas cuando crea necesario sin ningún tipo de gasto por su parte. Desde esta pantalla, el turista podrá ver los detalles, foto y situación de cada actividad siguiendo el procedimiento habitual ya comentado en los anteriores apartados.

Adicionalmente, se le da la opción de valorar las actividades que tiene dentro de la agenda. La votación que el usuario realice contará para la nota media que la actividad posee, a partir de todas las valoraciones que ha recibido, de esta manera podemos saber si la actividad ha gustado a otros turistas y hacernos una idea de si vale la pena o no.

Fig. 28 Agenda y votación

La votación realizada también puede afectar a nuestro perfil de usuario. En este caso hemos dado una nota de 1 (sobre 10) al museo de historia de Tarragona.

Una nota tan baja en una actividad con fuerte componente de historia nos produce una disminución de un punto en el campo interés en historia de nuestro perfil de usuario, disminución que tendrá en cuenta el recomendador, a partir de este momento, en las sucesivas peticiones que reciba por parte de este usuario.

10.9. Interfaz gráfica de los agentes de actividad

Los agentes especializados están situados en la parte del servidor y tienen su propia interfaz gráfica personalizada que mostraremos a continuación:

Agente Monumentos

Agente Itinerarios

Agente Exhibiciones

Agente Museos

Agente Conferencias

Agente Deportes

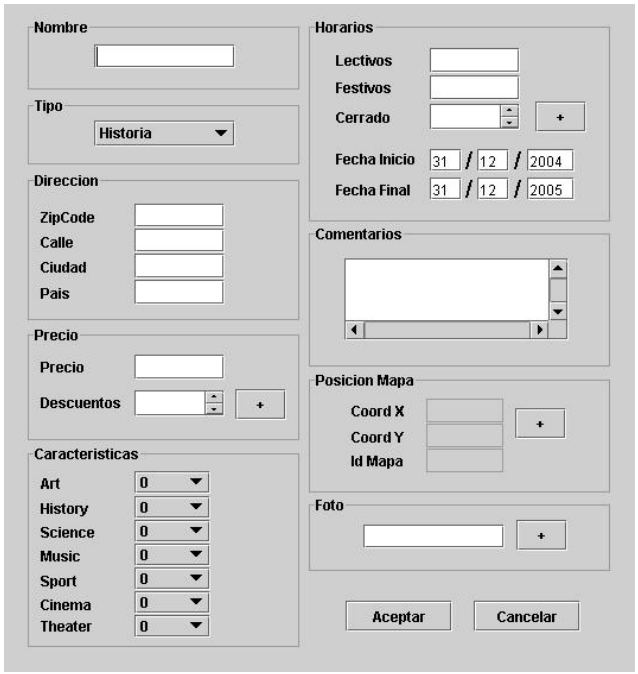
Agente Películas

Agente Teatros

Agente Conciertos

Todas estas interfaces poseen un botón “Add” para añadir nuevas actividades del tipo indicado. A continuación mostraremos el Dialog asociado a los “Adds” de dichas interfaces y también mostraremos la pantalla asociada a un Dialog Add donde se selecciona la localización de la actividad en la ciudad.

Dialog Add



Selecciona localización actividad

11. VALORACIÓN Y CONCLUSIONES

Los agentes son sin duda alguna uno de los campos principales donde la Inteligencia Artificial va a crecer más en un futuro próximo. La idea de tener varios agentes colaborando para resolver una tarea común que individualmente sería demasiado costosa, nos hace pensar que acabaremos aplicando estos novedosos sistemas en muchos y variados ámbitos. No obstante, no podemos olvidar que estamos hablando de una tecnología en su infancia; hay que ver cómo evoluciona y si definitivamente despega, en ámbitos mas cotidianos, puesto que en la actualidad es poco más que una herramienta de investigación.

Personalmente he de decir que este proyecto me entusiasmó desde el principio, por el reto que representaba implementar un asistente personal con unas importantes funcionalidades sobre un dispositivo móvil, que debía comunicarse utilizando GPRS con el resto del SMA. La investigación previa realizada fue sin duda uno de sus atractivos más importantes, sin olvidar los conocimientos adquiridos al adentrarnos en el uso del J2ME a nivel profesional.

Las mayores dificultades fueron encontradas en la parte de investigación previa, puesto que las limitaciones que nos imponía nuestro móvil y su software asociado, entorpecían una y otra vez nuestro desarrollo. La adaptación respecto a las versiones anteriores del SMA de turismo, también presento múltiples, y en algunos casos insalvables, problemas debido a las versiones de JADE utilizadas y a ciertas decisiones de diseño implementadas.

Los conocimientos adquiridos con la realización de este proyecto se pueden dividir en dos ámbitos.

En el entorno de la implementación he ampliado enormemente mi nivel de programación en Java y el acceso a bases de datos, obteniendo además unos excelentes conocimientos en J2ME y el uso del J2ME Wireless Toolkit de Sun. He obtenido un amplio conocimiento de JADE y LEAP aplicado al desarrollo de SMAs y ontologías asociadas. También quiero destacar el aprendizaje del entorno de programación Symbian utilizando C++ y las posibilidades de futuro que ofrece.

En el ámbito de la investigación realizada he obtenido un mayor conocimiento del entorno wireless, especialmente en cuanto a conexiones Bluetooth y GPRS se refiere. También he aprendido mucho respecto a los dispositivos móviles Nokia y su software/hardware asociado.

El hecho de haber trabajado en un proyecto con tecnología muy actual y con un gran futuro por delante, me hace ser optimista sobre lo que personalmente puedo aportar a las empresas interesadas en este ámbito, por ello estoy convencido de que me facilitará la tarea de buscar trabajo.

En cuanto a la incidencia de lo aprendido durante la carrera a la hora de realizar este proyecto, personalmente creo que todas las asignaturas cursadas me han resultado útiles, puesto que para realizar una investigación con éxito es indispensable poseer una densa base de conocimientos. No obstante de entre ellas podemos destacar *Redes de Computadores*, *Ingeniería del Software* e *Inteligencia Artificial*.

12. FUTUROS PROYECTOS

Hemos creado un SMA que ofrece un excelente servicio al turista, basándose en su eficiente funcionamiento en móviles vía GPRS, en su sistema de localización y en su módulo de recomendación de actividades.

Mejorar lo que tenemos, pasa por mejorar las tecnologías con que trabajamos. Estamos hablando de pasar de MIDP1.0 a MIDP2.0 con un aumento claro de la calidad de las funcionalidades ofrecidas por el aplicativo. De igual forma ello comportaría trabajar con un mejor dispositivo móvil que nos daría más prestaciones, mayor velocidad y por tanto mayor agilidad en nuestra aplicación.

El siguiente paso en la conectividad no puede ser otro que el UMTS. Con GPRS tenemos conexión donde quiera que exista cobertura telefónica para móviles, con UMTS obtendremos además ancho de banda y tarifa plana, conceptos indispensables al trabajar en un entorno distribuido que utiliza internet como enlace común.

El sistema de localización, basado actualmente en la red GSM, vería aumentada su precisión enormemente utilizando los modernos módulos GPS. Habría que estudiar hasta que punto las ventajas y los inconvenientes del GPS mejoran lo obtenido con el sistema GSM. No obstante sería una mejora a tener muy en cuenta.

El módulo de recomendación puede ser ampliado y mejorado, añadiendo y mejorando los sistemas especializados en recomendaciones diseñados por Montserrat Batet [Batet05] en su proyecto, aumentando así las funcionalidades ofrecidas, permitiéndonos una mayor eficiencia a la hora de recomendar actividades a los turistas.

Siguiendo la misma línea, el sistema de mapas debería mejorarse de forma que estos ocuparan menos espacio, agilizando la carga de dichos elementos por parte del móvil.

Por último, podríamos añadir nuevas funcionalidades al asistente personal, como por ejemplo un generador de rutas turísticas adaptadas al usuario.

13. AGRADECIMIENTOS

Para finalizar, me gustaría agradecer los recursos que me ha facilitado el GruSMA, indispensables para la correcta evolución del proyecto. Dentro del equipo del grupo de sistemas multiagente quiero destacar a David Isern por su apoyo, comentarios y pequeñas ayudas brindadas y muy especialmente a David Sánchez, que colaboró activamente con ideas y comentarios en varias fases del desarrollo del mismo y que sin su presencia, dicha labor hubiera sido mucho más complicada y costosa.

Mención especial se merece también Montserrat Batet, programadora del SMA de turismo en el cual nos hemos inspirado para la implementación de nuestro aplicativo. Su excelente trabajo y documentación ha facilitado en gran medida el desarrollo de nuestro proyecto.

Por último, agradecer también a todos aquellos que con sus comentarios, programas, documentos, pruebas, etc. hicieron posible la realización de este proyecto. Entre estos quiero destacar a Mika Raento [Raento05] estudiante de doctorado del *Helsinki University Department of Computer Science*, por sus explicaciones sobre la programación bajo el sistema operativo Symbian.

14. BIBLIOGRAFÍA

[Banzai05] Banzai, Grupo de investigación en Inteligencia Artificial, DEIM, Universitat Rovira i Virgili. http://www.etse.urv.es/recerca/banzai

[Batet05] Sistema multiagent de recomanació d'activitats turístiques i d'oci. PFC de Ingeniería en Informática de la Universidad Rovira y Virgili (URV). Marzo del 2005.

[Bluetooth05] Tecnología wireless para la conexión entre plataformas.
http://www.monografias.com/trabajos12/tecninal/tecninal.shtml
http://www.Bluetooth.com

[devicedb05] J2ME device Database. http://kissen.cs.uni-dortmund.de:8080/devicedb/index.html

[F.JavierGarcia05] Departamento de Ciencias de la Computación e I.A (DECSAI) de la Universidad de Granada.

[FIPA05] Foundation for Intelligent Physical Agents, http://www.fipa.org/

[FIPAandIEEE05] FIPA bajo la IEEE Computer Society, http://www.fipa.org/about/fipa_and_ieee.html

[FIPAprotocols98] Protocolos especificados por la FIPA, http://www.fipa.org/repository/ips.html

[ForumNokia05] Foro dedicado a los desarrolladores de software para móviles Nokia
http://www.forum.nokia.com

[Fujitsu05] Fabricante de hardware y software, http://www.fujitsu-siemens.es/

[Gprs05] Conectividad a Internet para móviles. http://www.gsmworld.com/technology/gprs/index.shtml

[GPS05] Sistema de localización global. http://www.gpsworld.com/gpsworld/

[GSM05] La red GSM. http://www.gsmworld.com/index.shtml

[GruSMA05] GruSMA, Grup de Sistemes Multi-Agent, DEIM, Universitat Rovira i Virgili, plana web,
http://www.etse.urv.es/recerca/rgai/toni/MAS/

[guide05] Documentos de apoyo al programador de entornos JADE.
http://JADE.cselt.it/doc/programmersguide.pdf

[Isern et al., 2001] Isern, D., Moreno, A., Valls, A., Desenvolupament de Sistemes MultiAgent en JADE, Report de Recerca, DEIM-RR-01-003, Universitat Rovira i Virgili, 2001

[JADE05] Plataforma para SMAs según la FIPA http://sharon.cselt.it

[JadeApi05] Api con las clases del entorno JADE http://JADE.cselt.it/doc/api/index.html

[Java05] Lenguaje programación orientado a objetos de Sun http://java.sun.com

[J2ME05] Java enfocado a la programación para móviles http://java.sun.com/j2me/

[KObjects05] Organización dedicada a la programación en j2me http://kobjects.sourceforge.net/

[L.JHerrera05] Departamento Ciencias de la Computación e I.A (DECSAI) .Universidad de Granada.
[LEAP02] Iniciadores del proyecto LEAP http://LEAP.crm-paris.com/

[mahmoud05] Professor at the Department of Computing and Information Science, University of Guelph. http://snowhite.cis.uoguelph.ca/~qmahmoud/

[mms.es05] Información general sobre móviles. http://www.mms.es

[Newlc05] Portal para desarrolladores de aplicaciones en entorno Symbian. http://www.newlc.com/

[Nokia05] Fabricante de móviles. http://www.nokia.es/index.jsp

[Pleumann05] Miembro de Kobjects y programador del midlet InfoSys.

[Raento05] Desarrollador de aplicaciones en entorno Symbian. http://www.cs.helsinki.fi/u/mraento/

[sma05] Explicaciones sobre agentes y sistemas multiagente http://agentes.ei.uvigo.es/

[specs05] Especificaciones de la FIPA http://www.fipa.org/specs/fipa00061/

[Sun05]. Sun Microsystems. http://www.sun.com/

[SunDev05] Sun Developers Network. http://developers.sun.com/

[Symbian05] Sistema operativo para dispositivos móviles. http://www.symbian.com/

[UMTS05] Conectividad a Internet para móviles. http://www.umtsworld.com/

[Wooldridge02] Wooldridge, M., *An introduction to multiagent systems*, John Wiley Ed., 2002. ISBN 0-471-49691-X.

[Zapata03] SMA turismo. PFC de Ingeniería Técnica en Informática de Sistemas de la Universidad Rovira y Virgili (URV). Septiembre del 2003.