UNIVERSIDAD PONTIFICIA DE SALAMANCA

Campus de Madrid

Facultad de Informática

DOCTORADO EN INGENIERÍA INFORMÁTICA

Programa en Ingeniería del Software

BIENIO 2003-2005

PROYECTO DE INVESTIGACIÓN:

PROPUESTA DE ARQUITECTURA PARA PLATAFORMAS DE ENTRETENIMIENTO DE TELEFONÍA MOVIL BASADA EN AGENTES

AUTOR:

César Hernán Parejas Llanovarced

Director:

Dr. D. Oscar Sanjuán Martínez

Madrid, Septiembre 2006

ÍNDICE

1. INTRODUCCIÓN A LA INVESTIGACIÓN

Los avances de las tecnologías de la información y las comunicaciones, y el auge actual de las comunicaciones inalámbricas ha dado lugar a la utilización cada vez más de dispositivos móviles con acceso a redes tales como PDA, Pocket Pc, Ordenadores Portátiles, Teléfonos Móviles, etc.

Por otro lado, en los últimos años el desarrollo Orientado a Agentes se ha convertido en un nuevo paradigma de Ingeniería del Software [AOSE 00], [MAS01]. El concepto de agente constituye una potente herramienta de abstracción en el desarrollo de software, que facilita la construcción de sistemas distribuidos, inteligentes y robustos [DMAT01].

Actualmente dentro del paradigma de la programación orientada a agentes están cobrando especial relevancia los agentes móviles [LANGE99], esto es, programas que pueden parar en cualquier momento, trasladarse a otros dispositivos y continuar su ejecución en el punto en que se suspendió lo cual, unido a las características propias de los agentes, permite llevar los cálculos al dispositivo en el que se encuentran los datos disminuyendo así el tráfico de información por una red. En este sentido, el lenguaje Java ha sido el que más éxito ha tenido para la implementación de infraestructuras de agentes.

El presente trabajo de investigación, pretende analizar el panorama actual de las plataformas de desarrollo de sistemas multiagentes orientados a los dispositivos móviles, además de presentar una propuesta de arquitectura de desarrollo orientada al entretenimiento para plataformas de telefonía móvil basada en agentes.

1.1 Justificación de la investigación

La telefonía móvil está liderando la transformación de las economías globales, cambiando la manera de trabajar y de vivir. Aunque ya cuenta con más de dos millardos de usuarios, el mercado de la telefonía móvil está lejos de saturarse. China, India, Rusia o Latinoamérica son algunos de los principales mercados que emergerán en los próximos años. Según la Guía Netsize [NEITSIZE06] el mercado de las comunicaciones inalámbricas se está configurando alrededor de cuatro áreas: negocio a través de dispositivos móviles *(mobile business)*, entretenimiento en el móvil *(mobile entertainment)*, comercio a través del móvil *(mobile commerce)* y marketing móvil *(mobile marketing)*.

El sector del entretenimiento en el móvil es el que más rápidamente ha crecido, especialmente debido a la descarga de música, vídeos, tonos y a los servicios de televisión interactiva. Por ejemplo, durante 2005 el negocio de la música en el móvil alcanzó en todo el mundo unos beneficios de 4,4 millardos de dólares.

La tecnología móvil también comporta grandes ventajas para las empresas, ya que ha permitido mejorar los procesos de negocio y la satisfacción de los consumidores. No obstante, para asegurar el crecimiento de este mercado es necesario, según Netsize [NEITSIZE06], que los fabricantes de sistemas operativos suministren una plataforma que permita desarrollar aplicaciones fáciles de usar sin necesidad de realizar grandes inversiones. Asimismo, a medida que el uso de estas aplicaciones se generalice los fabricantes deberán preocuparse por aumentar la seguridad de los dispositivos, que contienen información sensible, especialmente para las empresas.

Inicialmente concebido como un artículo de lujo sinónimo de un cierto estatus social, el teléfono móvil ha sido asumido por la población como un complemento imprescindible para la comunicación personal. Hoy su uso ya ha calado

profundamente en los segmentos más jóvenes, que entre otras cosas han provocado un inesperado éxito de los mensajes cortos (SMS) y conseguido una pequeña revolución en la estructura y distribución de los ingresos de las operadoras.

La telefonía móvil digital, la llamada segunda generación (2G), ha mejorado la calidad, eficiencia y seguridad de las comunicaciones móviles así como ha abaratado su uso permitiendo un masivo acceso. Además, ha hecho posible la estandarización del sistema GSM a nivel mundial, posibilitando el roaming entre países y operadoras, y un modelo de facturación al usuario final basado en acuerdos internacionales entre las operadoras.

El gran negocio que ha supuesto la telefonía móvil a operadores, fabricantes de terminales y equipos, distribuidores y proveedores de servicios, junto con la esperada integración con Internet ha forzado la investigación y desarrollo de nuevos estándares más eficientes. El objetivo principal es un mayor aprovechamiento del espacio radioeléctrico y, al mismo tiempo la introducción de nuevos servicios orientados a la transmisión multimedia y a las comunicaciones de datos. La finalidad de todo ello es garantizar la compatibilidad universal de las comunicaciones personales móviles.

Las nuevas tecnologías van a proporcionar un importante conjunto de herramientas para desarrollar todo tipo de servicios avanzados para los usuarios de la telefonía móvil. Entre estas tecnologías y servicios podemos destacar:

• Transmisión de datos de alta velocidad

Ya se puede utilizar el terminal móvil para navegar por Internet con una conexión de cierta velocidad. Con la tecnología GPRS, instalada en la mayor parte de Europa, se pueden obtener tasas de hasta 40 Kbps, comparables a las obtenidas con un modem normal sobre una línea telefónica fija.

Pero la tecnología 3G ya está aquí [UMTS00], ofertada comercialmente en la mayoría de operadoras, permite acceso de alta velocidad para todo tipo de aplicaciones de transmisión de datos. Estas velocidades llegan actualmente hasta los 384 Kbps pero en un futuro próximo se podrán elevar más allá de los 2 Mbps.

• Accesos multi-canal

SMS, el servicio de mensajes cortos, no va a ser nunca más la única tecnología de acceso a servicios móviles. MMS, y los sistemas de navegación ofertados por los operadores (imode, Vodafone-life, o e-moción) ya están disponibles para la descarga y utilización de una gran variedad de contenidos y servicios multimedia.

• Terminales multimedia

Los terminales han evolucionado para poder hacer todas estas aplicaciones y servicios disponibles. Son suficientemente potentes como para permitir explorar y acceder a todos los nuevos servicios y posibilidades. Su mejora se basa tanto en mayores capacidades (de red, de computación y memoria) como en ampliadas capacidades multimedia (pantallas de alta resolución a color, capacidades sonoras mejoradas, reproducción de los principales formatos de audio, video e imagen, así como cámaras de foto y video de alta resolución).

• Servicios multicontenido

Los servicios de descarga de tonos polifónicos, juegos y tonos a color ofrecen ahora más variedad que nunca.

1.1.1 El mercado de los juegos para móviles

Cada vez más los usuarios están interesados en formas móviles, portátiles de entretenimiento que ofrezcan nuevas posibilidades de juego. Las modernas tecnologías ya permiten todas estas posibilidades aunque hasta ahora no hayan sido aplicadas en todo su potencial.

Actualmente, hay una amplia gama de dispositivos electrónicos para juegos en el mercado, empezando por las consolas. La popularidad de las consolas de videojuegos está fuera de toda duda. Por ejemplo, Sony recientemente anunció ventas internacionales de más de 60 millones de consolas PlayStation 2 desde que el producto se lanzó al mercado hace algo más de 3 años. Los ordenadores personales son también otro medio popular para el juego electrónico, especialmente con el creciente aumento de actividades online.

Respecto a las consolas portátiles, la consola Game Boy Advance de Nintendo es el producto lider con más de un millón de consolas vendidas en Europa desde que salió a la venta. Más de 300 juegos de esta consola están ya en las estanterías de las tiendas y están preparados para su próxima distribución más de 50 nuevos.

Según IBM Business Consuting Services [IBM05], el número de usuarios de estos productos está también subiendo, especialmente entre las mujeres, personas por encima de los 30 años, familias y parejas. Casi el 20% de los jugadores está por encima de los 50 años, las mujeres con más de 18 conforman el segundo grupo de jugadores con el 26%, ganando al tradicional grupo de niños de 6 a 17 años que representan sólo el 21%.

En el mundo móvil la revolución de los juegos está empezando a ser un hecho. Ahora ya puedes descargarte juegos en tu teléfono, juegos a color que son mucho más sofisticados que los anteriores.

Hasta el 2003, los juegos eran principalmente instalados de fábrica en blanco y negro y de un solo jugador. Pero ahora, el desarrollo de juegos está en plena revolución y es un mercado con enorme potencial de crecimiento en los próximos años.

Esta revolución puede ser resumida con los siguientes avances:

- Desde los juegos en blanco y negro a los juegos a color

- Desde los juegos de un jugador a los juegos multi-jugador
- Desde los juegos embebidos a los juegos descargables

Estas mejoras sólo están siendo posibles gracias a la mejora de las capacidades de los terminales. Los usuarios ya no tendrán nunca más que jugar con juegos en blanco y negro monojugador, o preinstalados en su terminal. Ahora podrán fácilmente descargar juegos a color e incluso jugar contra usuarios conectados a la misma red.

Aún así, consideramos que existen tres principales obstáculos para ofrecer estos servicios de entretenimiento móvil:

- Primero, ofrecer un servicio de alta calidad que sea capaz de gestionar cómodamente una gran cantidad de peticiones de descarga en cortos espacios de tiempo y que permite realizar rápidas descargas.

- Segundo, desarrollar juegos interactivos con funciones para multi-sesión, multi-jugador manteniendo interfaces de usuario atractivos y amigables.

- Finalmente, proveer todos estos servicios a pesar de las numerosas y complejas diferencias que existen entre las capacidades de los distintos modelos de terminales móviles.

Centrándonos en España, los juegos son el servicio móvil avanzado más utilizado tal y como podemos ver en el siguiente gráfico, por delante de la mensajería y la descarga de tonos y logos para el móvil.

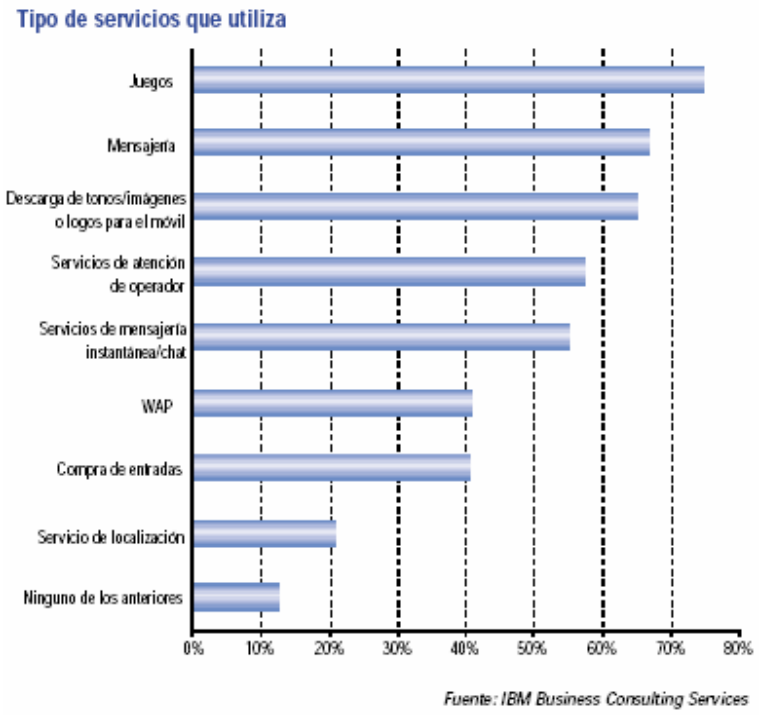

Figura 1.1: Tipo de servicios utilizados por los usuarios en el año 2005

Y lo que es más importante, analizando las respuestas del colectivo de los más jóvenes, a las preguntas de que servicios móviles estarían dispuesto a usar en un futuro, también son los juegos la primera opción, por lo que en pocos años los juegos móviles en España serán uno de las aplicaciones móviles más demandadas, como ya sucede actualmente con los videojuegos, tanto mediante consolas, como mediante ordenadores personales.

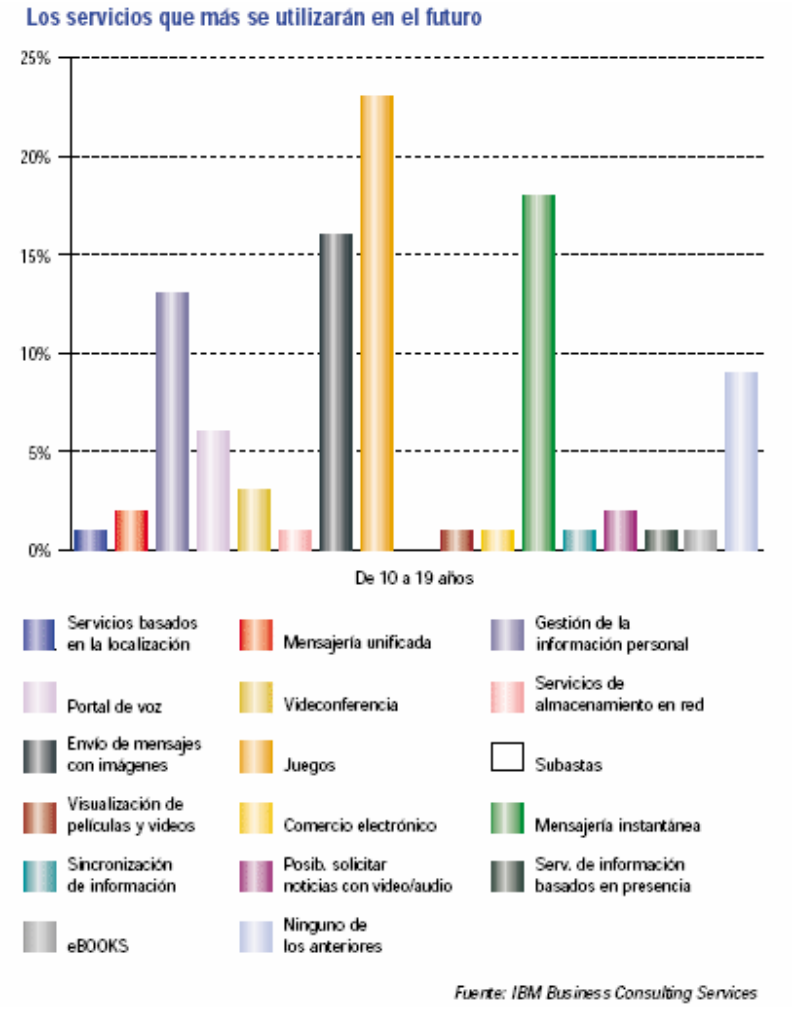

Figura 1.2: Los servicios que se utilizarán en el futuro

1.2 Hipótesis y punto de partida

El uso de la arquitectura propuesta para el desarrollo de agentes orientado al entretenimiento bajo la plataforma Jade-Leap [LEAP02], mejora las prestaciones de los juegos multi-jugador interactivos de tiempo real para terminales móviles.

1.3 Objetivo

Contar con plataformas que permitan la implementación de agentes personales en dispositivos móviles, tales como JADE-LEAP, es esencial para un próximo

despliegue masivo de servicios bajo demanda de agentes y de su aceptación social.

El presente trabajo de investigación, tiene como objetivo demostrar cómo se puede utilizar una arquitectura basada en JADE-LEAP para implementar un agente de entretenimiento personal en un terminal de telefonía móvil. Este agente forma parte de un sistema multi-agente que permitirá al usuario jugar de manera interactiva en tiempo real con otros usuarios mediante su teléfono móvil. El agente personal se comunica de forma inalámbrica, con el resto de los agentes del sistema multi-agente, para acceder a la partida más apropiada en la que pueda jugar el usuario.

1.4 Metodología de la investigación

La metodología a utilizar, está basada en el método científico y consta de las siguientes etapas:

1. Formulación del problema

2. Exploración:

    a) Búsqueda de información.
    b) Construcción del marco teórico.

3. Diseño de la investigación

4. Desarrollo de la investigación y estado del arte

5. Caso práctico

6. Conclusiones

La formulación del problema se verá plasmada en el apartado de objetivos, mientras que la fase de exploración y el diseño de la investigación se desarrollan durante la hipótesis y punto de partida de la investigación.

La fase de investigación se realiza en el apartado de situación y estado del arte, donde se estudian la situación actual del desarrollo de agentes en dispositivos móviles y el estado del arte de los videojuegos.

El planteamiento de la arquitectura para el desarrollo de agentes de entretenimiento bajo la plataforma Jade-Leap, es el centro de este trabajo de investigación. El caso práctico de estudio refuerza nuestro planteamiento, así como sienta las bases de futuras investigaciones.

1.5 Organización del trabajo

Este trabajo está organizado en 6 capítulos según la siguiente estructura:

En el primer capítulo (introducción) se presenta brevemente una reseña sobre la investigación, hipótesis, su justificación y la descripción de la estructura de la memoria.

En el segundo capítulo (Tecnología de Agentes móviles) describiremos brevemente los aspectos más destacables de la tecnología de agentes móviles, así como las características que las hacen interesantes frente a soluciones basadas en una aproximación cliente/servidor.

En el tercer capítulo (Estado del arte: situación actual del desarrollo de aplicaciones multiagentes para dispositivos móviles) analizamos el panorama actual de la tecnología de agentes móviles aplicado a los dispositivos móviles de reducidas prestaciones. Estudiamos los inconvenientes y ventajas del uso de la

tecnología de agentes en este escenario, así como las soluciones planteadas por la comunidad científica a día de hoy.

El cuarto capítulo (Estado del arte: situación actual del desarrollo de juegos para dispositivos móviles) describe en principio de forma breve la clasificación de juegos que fueron surgiendo desde el Space War, considerado el primer juego diseñado por ordenador el año 1962, hasta los actuales juegos en 3D; analizando los tipos de juegos que son más apropiados para un terminal móvil, y cuales serían inviables. En la segunda parte de este capítulo, analizamos el mercado actual de los terminales móviles de la serie 60 de Nokia (http://www.series60.com), elegida por sus prestaciones técnicas que se adecuan mejor para la consecución de nuestros objetivos.

En el quinto capítulo (Propuesta de arquitectura de desarrollo de juegos persistentes multigujador en la plataforma de agentes jade-leap para terminales móviles) presentamos nuestra propuesta de arquitectura que facilita el desarrollo de aplicaciones lúdicas multijugador con agentes móviles personalizados. Para ello utilizamos como plataforma base de desarrollo de agentes, la plataforma JADE-LEAP [LEAP06]. En la segunda parte de este capítulo, presentamos un caso de estudio práctico para evaluar la viabilidad de nuestra propuesta.

En el capítulo sexto se presentan las conclusiones finales del trabajo de investigación. Para ello se evalúan los objetivos alcanzados a partir de la hipótesis de partida, comprobando el cumplimiento de las expectativas iniciales.

Al final de la memoria se presentan los anexos, conformado en este caso por la bibliografía y referencias que empleamos como fuente de investigación y un glosario de términos con la finalidad de facilitar la comprensión de la lectura del trabajo.

2. TECNOLOGÍA DE AGENTES MOVILES

2.1 Generalidades

La tecnología de agentes móviles ha tenido un especial impacto en los últimos años gracias a su aplicación en la computación distribuida. La ventaja que proporciona respecto a los modelos clásicos cliente/servidor es que permite realizar las mismas operaciones pero con la ventaja de que son asíncronas y que además no se precisan conexiones permanentes para la ejecución de tareas, puesto que el agente que migra hacia otro sistema además de llevar los datos necesarios para realizar la operación, conserva la información de estado del proceso.

Aunque el término agente no es un término nuevo, no hay todavía una definición estandarizada y aceptada por todos. A la hora de definir un agente, existe una clara controversia entre los investigadores en función de sus características. Sin embargo, sí existe un consenso en el sentido de que es una entidad con capacidad de representación y con una serie de atributos, características o propiedades fundamentales que tienen que tener todos los agentes, amén de unos atributos adicionales que aumenten su funcionalidad [CORCHADO02]:

- Atributos fundamentales:
  - ✓ Independencia de actuación, es decir, sin sufrir la intervención directa de nadie.
  - ✓ Flexibilidad:
    - Reactividad-Adaptabilidad: Perciben, se adaptan al entorno y actúan en él.
    - Proactividad: Capacidad de tomar la iniciativa.
    - Comunicabilidad-Sociabilidad: Acceso a recursos y capacidad de interacción con otras entidades.
  - ✓ Continuidad Temporal: Proceso en continua ejecución.

- Atributos adicionales:
  - ✓ Movilidad (Agentes móviles): Migrar, interactuar y regresar bajo su propio control.
  - ✓ Capacidad de Razonamiento/Aprendizaje: Inteligencia.
  - ✓ Seguridad/Confiabilidad: Mecanismos y servicios de seguridad para evitar sorpresas por parte del agente representante y/o de otras entidades.

Según se muestra en la siguiente Figura 2.1, la definición más completa de un agente sería aquélla que junte los atributos básicos de la visión amplia con los atributos opcionales de la visión estricta. En definitiva, esta definición se correspondería con todos los atributos que en teoría debe disponer un agente móvil, inteligente y seguro.

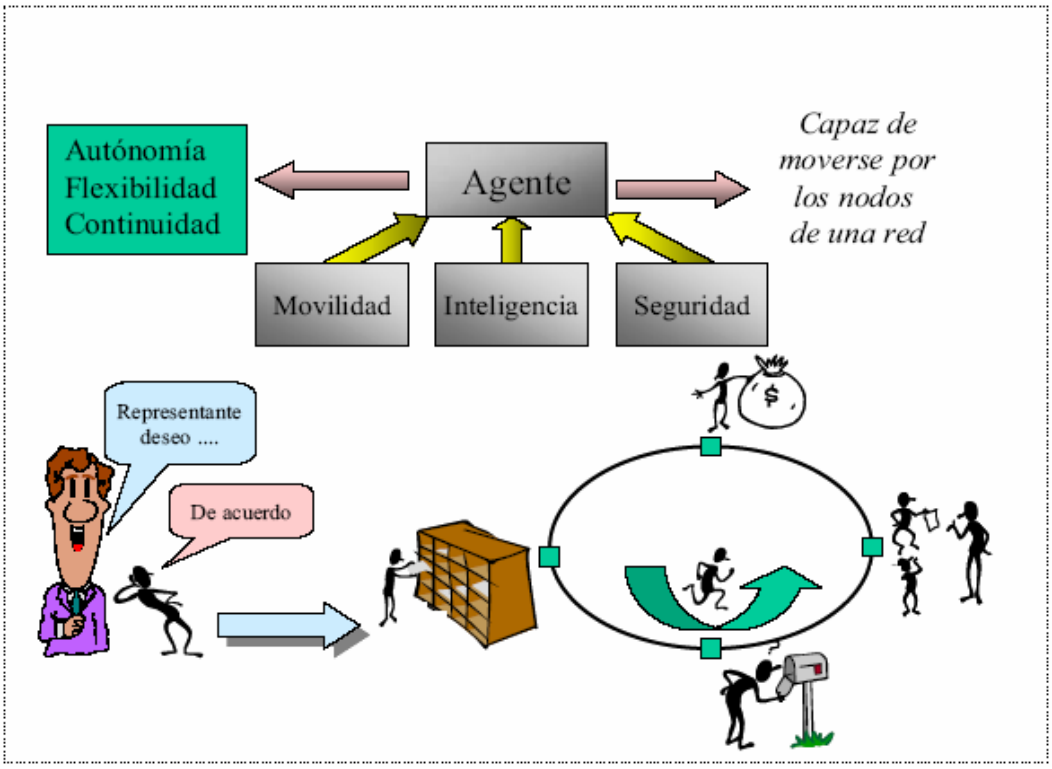

Figura 2.1. Lo ideal: un agente móvil, inteligente y seguro

Si la definición de un agente es algo compleja, su clasificación también lo es ya que la taxonomía de los agentes es muy amplia y variada. Para una mayor comprensión, clasificaremos a los agentes de software en dos grandes tipos y con las siguientes características [JENNINGS02]:

- Agentes estáticos:
  - ✓ Ejecución limitada al sistema donde se inicia.
  - ✓ Interactúan con entidades locales: agentes, programas, usuarios.
  - ✓ Accesos a recursos remotos vía sockets[1], RPC[2], RMI[3], CORBA[4], etc.

- Agentes móviles:
  - ✓ Capacidad de movimiento: Migración.
  - ✓ Interacción directa con entidades remotas: Programación Remota.
  - ✓ Transportan un mensaje: Estado (datos) y código (proceso o clase).

Por consiguiente, un agente estático es un proceso que se ejecuta única y exclusivamente en la máquina donde comienza la ejecución. Si necesita información que no está en el sistema o necesita interactuar con otro agente en una máquina diferente; utiliza, por ejemplo, un mecanismo de comunicación vía sockets[1], RPC[2], RMI[3], CORBA[4], etc. Por el contrario, un agente móvil es un proceso que no está limitado al sistema donde se inicia la ejecución sino que tiene la completa libertad para moverse por la red, migrando, interactuando y regresando bajo su propio control. A su vez, la migración consiste en suspender la ejecución, transportarse a otra plataforma en la red y reanudar la ejecución en el punto en donde se suspendió.

---

1 Sockets: concepto abstracto por el cual dos programas (situados en computadoras distintas) pueden intercambiarse cualquier flujo de datos, generalmente de manera fiable y ordenada.

2 RPC: (*Remote Procedure Call*): protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

3 **RMI** (**Remote Method Invocation):** Mecanismo de invocación remota de Java

4 **CORBA (Common Object Request Broker Architecture):** Especificación de la OMG para la construcción de plataformas distribuidas independientes del lenguaje de programación.

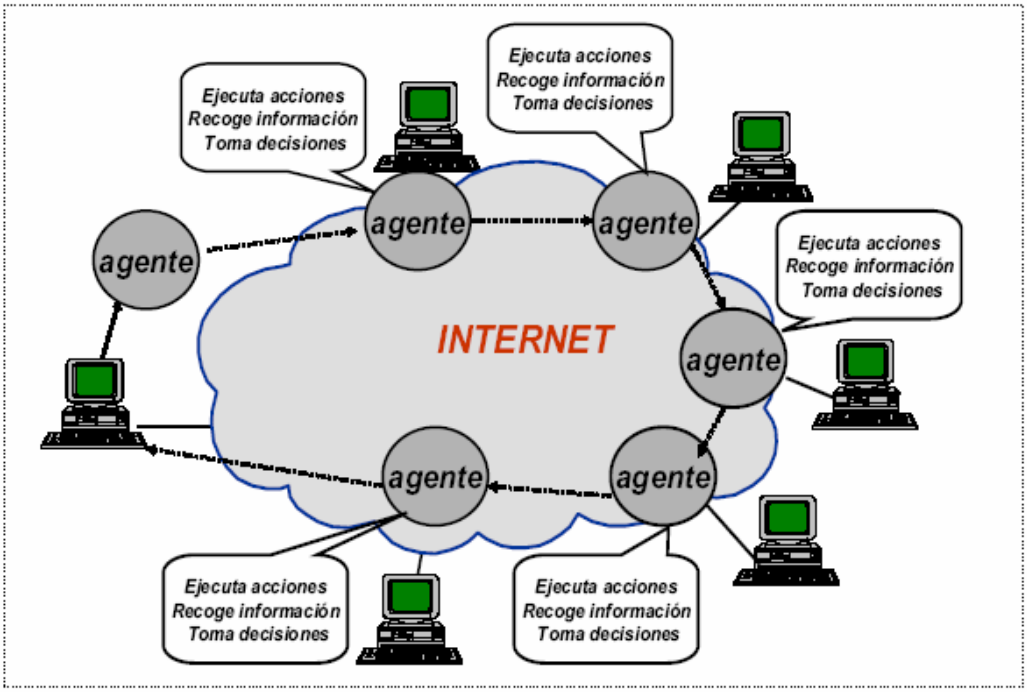

Figura 2.2. Un agente móvil viajando por la red

En concreto, un agente móvil es un proceso situado en una plataforma o entorno de ejecución para actuar en representación básicamente de una persona u organización, y llevar a cabo una tarea para la cual ha sido especialmente designado. Asimismo, dispone de un conjunto más o menos amplio de atributos entre los que destaca la movilidad en el sentido de que no está limitado al sistema donde comienza la ejecución sino que tiene la completa libertad para viajar, es decir, migrar, interactuar y regresar bajo su propio control.

Antes de continuar, conviene distinguir entre código móvil y ejecución remota de lo que se entiende por agente móvil y migración. Un código móvil tiene una sola vida, es decir, se transfiere una sola vez antes de su ejecución, la cual se efectúa de forma completa desde su inicio hasta el final. En este contexto, un agente móvil tiene diferentes vidas o “resurrecciones” en el sentido de que puede moverse por un itinerario de entornos de ejecución interactuando con otros agentes o recursos de información y computación hasta llevar a cabo el servicio solicitado; momento

en el cual, regresa a la máquina de partida. Como ejemplos de código móvil se pueden citar los siguientes:

- ***Applet (Sun Microsystems).-*** Es una aplicación Java compilada previamente y cuyo código objeto (bytecode) se referencia en una página HTML dentro de un servidor Web. Posteriormente, dicho código objeto se descarga por la red para su interpretación (intérprete Java) y ejecución en un cliente Web.

- ***JavaScript (Netscape).-*** Es una pequeña aplicación o script hecha en un lenguaje interpretado (lenguaje JavaScript) basado en comandos y cuyo código fuente se referencia en una página HTML dentro de un servidor Web. Seguidamente, dicho código fuente se descarga por la red para su interpretación (intérprete JavaScript) y ejecución en un cliente Web.

- ***Controles ActiveX (Microsoft).-*** Es una aplicación hecha en un lenguaje cualquiera que se compila previamente y cuyo código objeto se referencia en una página HTML dentro de un servidor Web. Posteriormente, dicho código objeto se descarga por la red para su ejecución directa (sin intérprete) en un cliente Web.

Como se muestra en la Figura 2.3, los agentes móviles son funcionalmente objetos de software que transportan en un mensaje su estado y código para poder interactuar remotamente con otros agentes o recursos [GIANP01]. Para que el agente móvil pueda interactuar con otros agentes o recursos afines es necesario la definición de un protocolo que especifique el formato de los mensajes y las acciones que se han de efectuar. En el mensaje transportado por el agente móvil se incluye un:

- Estado: Contiene el estado del objeto o los valores de los atributos del agente y el estado de la máquina o estado de ejecución (contador de programa, punteros de pila, registros, ...).

- Código: Incluye la clase con los métodos de interacción permitidos.

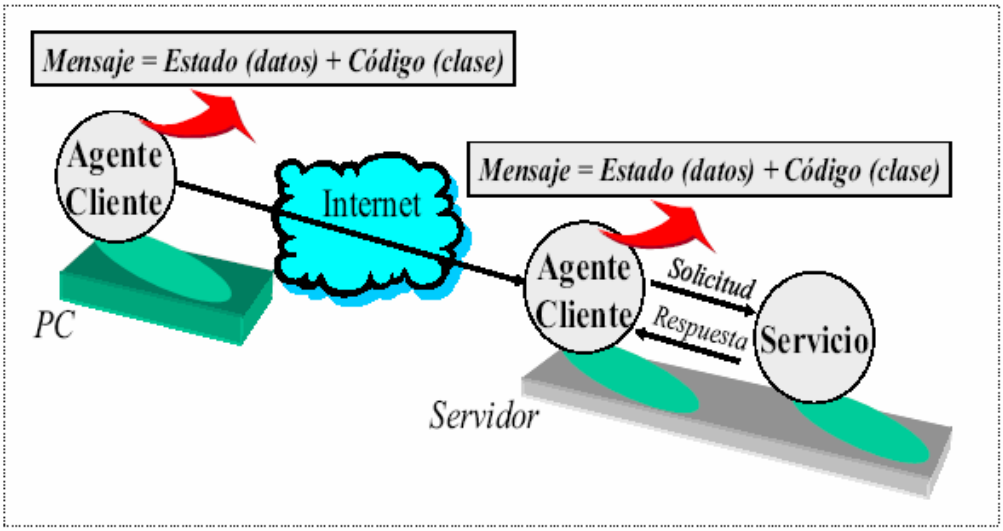

Figura 2.3. El mensaje transportado por un agente móvil.

Como se ha visto, cuando un agente se mueve, realiza una migración suspendiendo la ejecución, transportándose a otra plataforma en la red y reanudando la ejecución en el punto en que se suspendió. En este contexto, existen dos tipos de migración que se diferencian por el transporte de más o menos información o datos de estado [JENNINGS02]:

- **Migración fuerte:** Transporta el estado del objeto (valores de los atributos del agente) y estado de la máquina o estado de ejecución (contador de programa, punteros de pila, registros, ...). Es la "foto completa", justo cuando se suspende la ejecución para su posterior reanudación en el entorno de otra máquina.

- **Migración débil:** Transporta sólo el estado del objeto (valores de los atributos del agente). Es la "foto incompleta" que permite sólo reiniciar la ejecución con los valores actuales de los atributos. Por otro lado, es la típica migración utilizada por los agentes móviles escritos en Java, debido a que este lenguaje no permite la captura del estado de ejecución de un hilo de Java. Se resalta que el modelo de seguridad de la arquitectura JVM (Java Virtual Machine)

impide que un programa acceda o manipule directamente sus punteros, registros y pilas. Por consiguiente, cuando un agente Java se transporta de una plataforma a otra, no reanuda su ejecución sino que reinicia ésta con los valores actuales de sus atributos. Evidentemente, para que un agente Java reanudara su ejecución en otra plataforma sería necesario que el programador modificara internamente la máquina JVM realizando las oportunas extensiones o bien simulara el estado de ejecución codificando éste con una serie de variables del programa y un método de arranque.

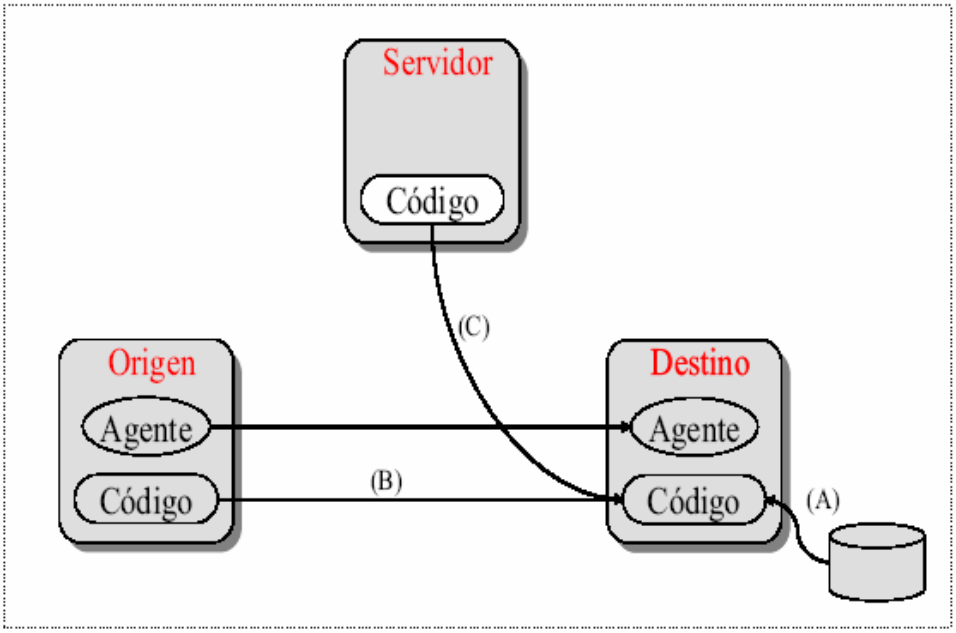

Figura 2.4. La transferencia del código de la clase de un agente.

Con respecto al transporte de la otra parte del mensaje de un agente móvil, es decir, el código de la clase, según se muestra en la Figura 2.4, se presentan tres posibilidades:

A. El código se encuentra disponible en la plataforma de destino ya sea en la memoria caché de clases o en el sistema local de ficheros. Consecuentemente, no es necesario transportar dicho código en el mensaje del agente.

B. El código se encuentre disponible sólo en la plataforma de origen. Este es el caso más frecuente y, por tanto, es necesario su transferencia en el mensaje del agente con la lógica penalización del tráfico de la red.

C. El código se encuentre en un servidor de clases. Se procede a recuperarlo bajo demanda desde la plataforma de destino.

Llegados a este punto, es importante distinguir ahora entre objetos y agentes móviles. Aunque los agentes son funcionalmente objetos, existen, entre otras, las siguientes diferencias [HUHNS97]:

- Un objeto no es un agente móvil ya que un agente móvil es más que un objeto, es decir, es autónomo, flexible, etc.

- Un objeto no tiene autonomía sobre su comportamiento a la hora de ejecutar o no un determinado método.

- Un objeto no es flexible, es decir, no combina un comportamiento reactivo, proactivo y sociable.

- Desde un agente móvil no se invoca un método sino que se realiza una solicitud de servicio que el agente llevará a cabo o no.

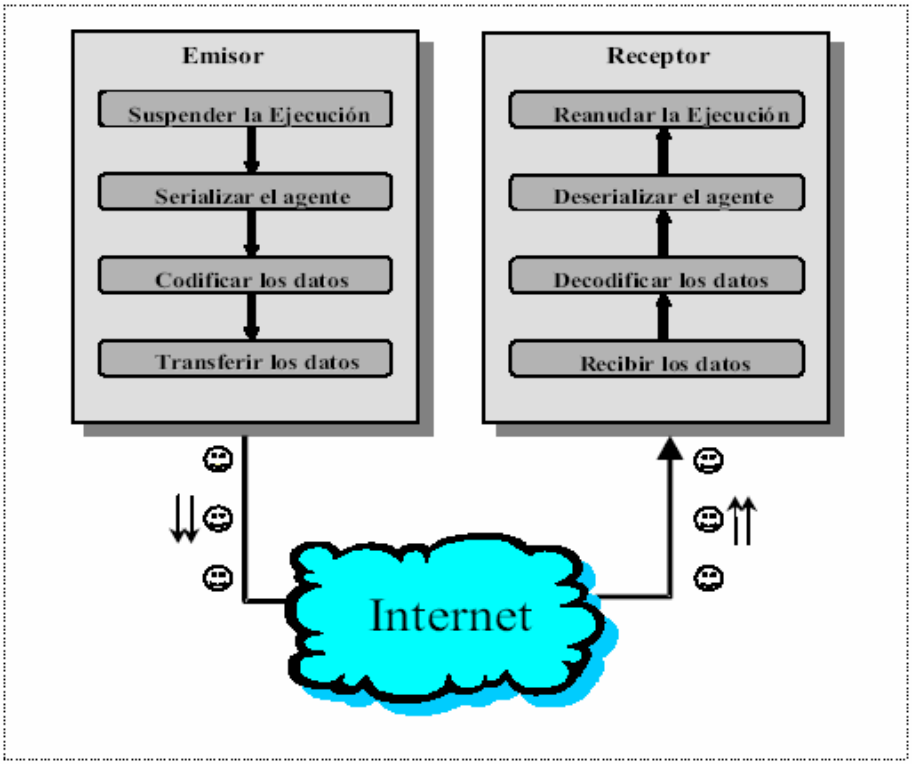

Figura 2.5. Transferencia de un agente móvil.

En la Figura 2.5 se muestran las distintas acciones que hay que realizar con un agente móvil para que éste pueda migrar desde un entorno de ejecución a otro [HUHNS97]. La migración de un agente consiste básicamente en suspender la ejecución, transportarse a otra plataforma en la red y reanudar la ejecución en el punto en que se suspendió. Así en el lado del emisor (plataforma de origen) se suspende la ejecución y se serializa al agente, es decir, se serializa su estado y código en un determinado formato de serialización o de transferencia, es decir, en un flujo de octetos (stream) transferible capaz de ser transportado por la red y restaurado en el lado receptor (plataforma de destino). Seguidamente, se efectúa el marshalling [W3C06], es decir, se toman los datos serializados y se codifican en un formato o sintaxis común de codificación, representación y transferencia. Finalmente, se transmite al agente, por ejemplo, vía sockets, RPC, Java RMI, CORBA, etc. A su vez, en el lado del receptor (plataforma de destino) se realiza el proceso contrario o unmarshalling, es decir, se decodifican los datos serializados para luego deserializar los octetos de dichos datos al formato nativo de la nueva

máquina; con el objetivo final de poder reiniciar (migración débil) o reanudar (migración fuerte) la pertinente ejecución.

2.2 Escenarios de aplicación de los Agentes móviles

La solución de problemas complejos no se lleva a cabo por un único agente, sino que es necesario un conjunto de agentes que interactúan entre ellos para la consecución de los objetivos del sistema. La utilización de varios agentes representa de forma natural la descentralización del sistema y enfatiza la comunicación para gestionar las dependencias entre ellos.

Entre los diversos escenarios de aplicación donde se pueden considerar actualmente el uso de agentes móviles, tenemos:

- Recopilación de información de estado:
  - ✓ El agente móvil recoge información diseminada por la red.
  - ✓ Se establece un itinerario para que un agente móvil viaje secuencialmente por una red de área local recopilando, por ejemplo, información sobre el estado de almacenamiento de los discos duros de las máquinas servidoras de la organización.
  - ✓ Ejemplo de un proyecto de investigación en este escenario es el “Sistema de acceso a Información Personal desde Entornos con conectividad limitada” de la Universidad de Vigo [SAIPE01].

- Búsqueda y Filtrado de Información:
  - ✓ El agente móvil parte de un conocimiento de las preferencias del usuario y elimina la información irrelevante.
  - ✓ Ejemplo de una aplicación en este escenario es el de la búsqueda de servicios de taxi [Moreno02].

- Monitorización de Noticias:
  - ✓ El agente móvil se envía para que esté a la escucha de ciertos tipos de información y recoja una determinada información diseminada por la red a través del tiempo

- Copia o mirroring:
  - ✓ El agente móvil realiza una copia inteligente de una información que se actualiza a determinadas horas del día o la noche.

- Diseminación de la Información:
  - ✓ El agente móvil lleva a cabo una distribución interactiva de noticias o publicidad a grupos interesados.

- Negociación:
  - ✓ Los agentes móviles negociadores efectúan una planificación de reuniones en nombre de sus representados.

- Comercio Electrónico:
  - ✓ El agente móvil localiza productos y precios para, finalmente, llevar a cabo una comparación, negociación y compra.

- Entretenimiento:
  - ✓ El agente móvil se programa con una determinada estrategia para su posterior envío a un servidor de juegos. Opcionalmente, se puede establecer todo un itinerario de servidores de juegos.

- Subastas:
  - ✓ El agente móvil se programa con una determinada estrategia para su posterior envío a un servidor de subastas. Opcionalmente, se puede establecer todo un itinerario de servidores de subastas.

2.3 Agentes móviles: una alternativa para la computación distribuida

Antes de que la tecnología de agentes móviles se hiciese realidad, sólo existía una aproximación para el diseño de sistemas distribuidos: la aproximación cliente/servidor, donde un módulo denominado "cliente" invoca servicios remotos de un módulo remoto que acepta dichas llamadas, denominado "servidor". Este procedimiento también se denomina llamada a procedimientos remotos o *Remote Procedure Calling (RPC).* Existen diferentes protocolos de comunicaciones que implementan esta idea, como RMI y CORBA.

Sin embargo, con la llegada de los agentes móviles existe una nueva posibilidad. Los agentes móviles son módulos inteligentes y autónomos que se mueven por sí mismos de un ordenador a otro, llevando consigo su estado y continuando su ejecución en el ordenador remoto. Algunas ventajas del uso de los agentes móviles, relacionadas con el acceso a información remota y la capacidad de interconexión, son los siguientes:

- *Encapsulan el protocolo de comunicaciones.* El agente móvil sabe cómo moverse por si mismo de un lugar a otro, no hay código de transferencia de información entre ordenadores.

- *Son asíncronos.* No necesitan comunicaciones síncronas para trabajar, dada su naturaleza autónoma, aunque por supuesto pueden sincronizarse con otros módulos o agentes.

- *Permiten reducir el uso de la red.* En el modelo RPC clásico, cuando un servicio es invocado remotamente, la conexión de la red debe estar abierta desde la invocación del método remoto hasta que se obtienen los resultados. En el caso de los agentes móviles, cuando el agente ha llegado al ordenador destino, la conexión de red ya no se necesitará hasta que el agente necesite viajar de nuevo o comunicarse remotamente.

- *Interacción local.* En lugar de realizar llamadas a procedimiento remotos, los agentes pueden viajar al ordenador adecuado e interactuar localmente con el sistema servidor lo cual puede ser crítico en aplicaciones como las basadas en tiempo real.

- *Adaptabilidad al contexto.* Pueden comportarse de forma diferente en sitios diferentes, adaptándose a los recursos disponibles.

2.4 Sistemas de agentes (Agencias o Plataformas de agentes): Introducción

Según Nicholas Jennings [JENNINGS98], un sistema de agentes (SA) o agencia es una plataforma o entorno de ejecución de agentes que permite, fundamentalmente, la creación, suspensión, terminación y ejecución inicial de un agente o retomar su ejecución, posibilitando la comunicación entre agentes y sus transferencias. Por tanto, un sistema de agentes es responsable de efectuar las siguientes funciones:

Figura 2.6: Ejemplo de un sistema de agentes en una máquina

- ✓ Coordinar todas las acciones entre agentes.

- ✓ Garantizar la seguridad del sistema interactuando con otras agencias y evitando la llegada de agentes procedentes de agencias maliciosas.

- ✓ Permitir la asociación con un representado (autoridad) que identifique a una persona u organización.

- ✓ Disponer de un tipo que, a su vez, describa parte del perfil de un agente. El perfil completo de un agente queda definido por los tres siguientes parámetros: Tipo del SA, lenguaje y método de serialización.

- ✓ Disponer de uno o varios lugares (entornos específicos de ejecución) en función de los intereses y objetivos de los agentes). Como muestran la Figura 2.7 y Figura 2.8, en cada lugar se pueden ejecutar uno o más agentes y, a su vez, en una máquina se pueden alojar simultáneamente uno o más SA. Si un lugar dispone de más de un agente, cada uno de ellos ofrecerá una clase de servicio diferente para ese mismo lugar. Así, se podría disponer de un lugar de comercio electrónico con distintos agentes especializados en la localización, comparación y compra de distintos productos. Otro lugar de ocio con uno o más agentes con distintas estrategias para enviarlos a servidores de juego, etc.

- ✓ Disponer de un servicio de nombres que permita encontrar agentes y lugares locales. En este contexto, cada agente dispone de un identificador global y único formado por la concatenación de las siguientes informaciones: Autoridad representada, SA, lugar, un valor o identificador y la dirección IP de la máquina.

- ✓ Disponer de una infraestructura de comunicaciones (IC) que ofrezca los pertinentes servicios de transporte entre los SA.

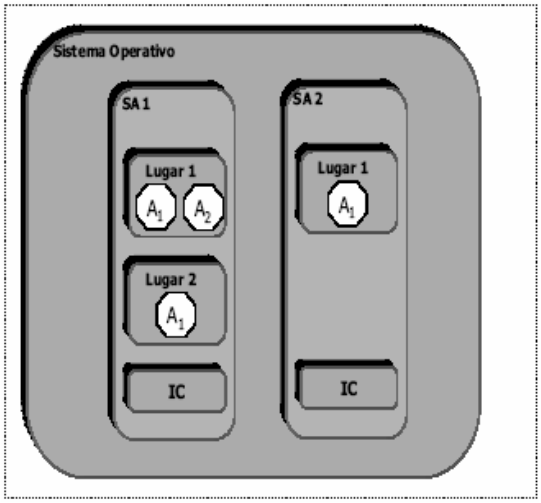

Figura 2.7. Un ejemplo de dos SA en una misma máquina.

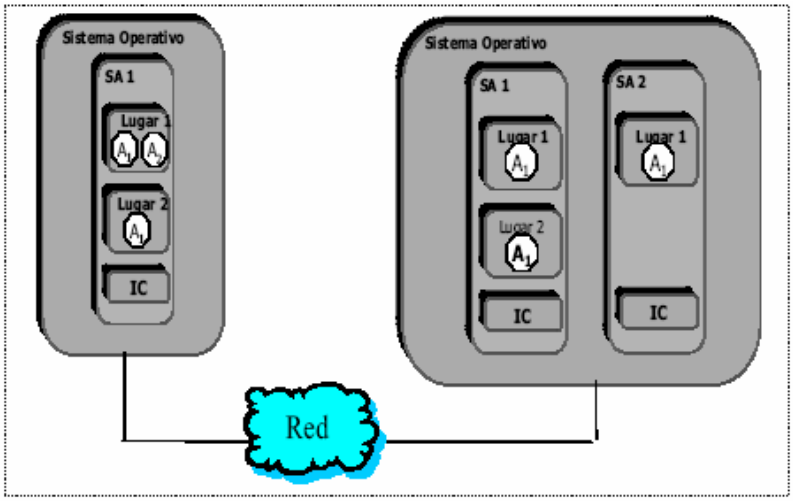

Figura 2.8 Ejemplo de interconexión de los SA de dos máquinas.

El hecho de que los agentes móviles no actúan de forma aislada se puede ver todavía de una forma más clara a través del concepto de región [LUCK03] que permite a una autoridad estar representada por uno o más SA (ver fig. 2.9 y fig. 2.10). Una región contiene uno o más SA con la misma autoridad, pero no necesariamente con el mismo tipo de sistema de agente. Asimismo, una región es responsable de llevar a cabo las siguientes funciones:

- ✓ Facilitar la gestión de los componentes distribuidos (los SA, lugares, agentes).

- ✓ Mejorar la escalabilidad (mejor distribución de carga a través de múltiples SA).

- ✓ Mejorar el control de acceso y la seguridad.

- ✓ Disponer de un servicio de nombres al cual se conectan los servicios de nombres de cada SA.

- ✓ Interconectarse con otras regiones a través de los llamados puntos de acceso a la región (SA externas), los cuales pueden compartir un mismo servicio de nombres (previo acuerdo entre sus autoridades). Un punto de acceso a la región es otro SA que mantiene información de todos los SA de la región.

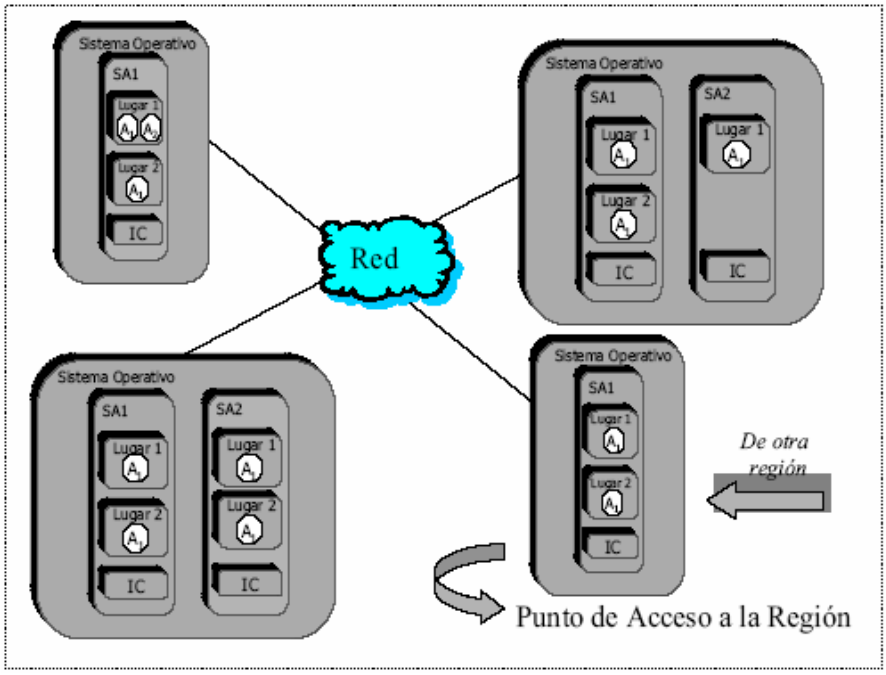

Figura 2.9. Ejemplo de región.

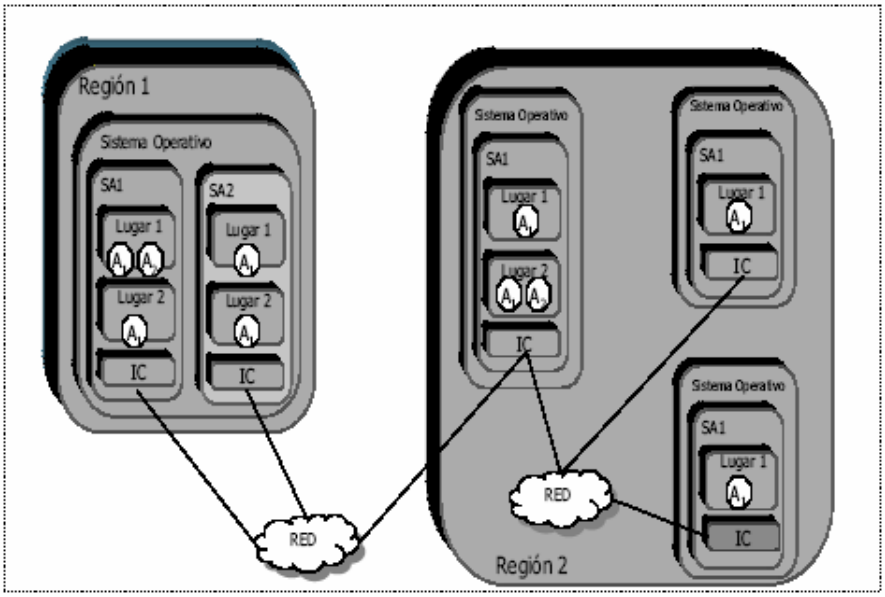

Figura 2.10. Ejemplo de interconexión entre dos regiones.

En la siguiente Figura 2.11, se describen los pasos más significativos que debe realizar un agente móvil "A" para interactuar con otro remoto "B".

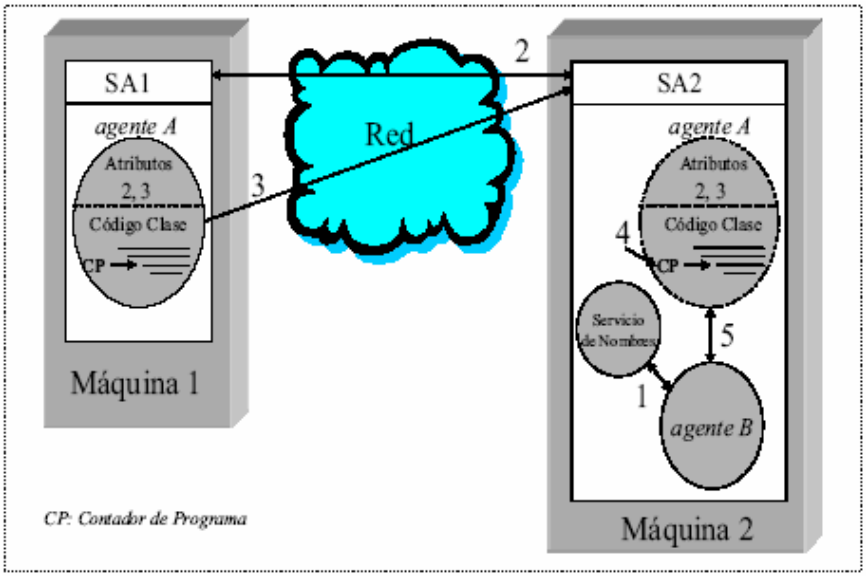

Figura 2.11. Ejemplo de cómo un agente solicita interactuar con otro.

1. Inicialmente, el agente "B", en el SA2, se conecta al propio servicio de nombres de SA2 para registrarse. El agente "A" podría conocer al agente "B" por el servicio

de nombres de la región al cual se conectan los servicios de nombres de cada sistema de agentes.

2. Previamente, el agente “A” interrumpe su ejecución ante el deseo de interactuar con el agente “B”. A continuación, transmite a SA1 una solicitud de permiso para viajar a SA2. Finalmente, SA1 solicita una autorización a SA2 para poderle enviar al agente “A”. En función de la identidad, credenciales y privilegios de la región, SA y autoridad, se emitirá o no el pertinente ¡OK!

3. Seguidamente, el agente “A” migra con la identidad, credenciales y privilegios de la región, SA y autoridad. Un agente debe disponer de un identificador global y único formado por la concatenación de las siguientes informaciones: Autoridad representada, región (si existe), SA, lugar, un valor o identificador y la dirección IP de la máquina.

4. El agente “A” retoma su ejecución en SA2 a partir de su información de estado.

5. El agente “A” interactúa con el agente “B”.

2.4.1 Lenguajes de programación para el desarrollo de sistemas de agentes

En un principio se puede utilizar cualquier lenguaje de programación aunque no esté específicamente diseñado para el desarrollo de los agentes móviles y sus plataformas o entornos de ejecución. Este tipo de lenguajes obliga al programador a desarrollar al agente móvil desde cero teniendo que incluir aquellas funciones, operaciones y comandos que soporten las características de la teoría de agentes que desee añadir. En este contexto, se destacan los siguientes lenguajes:

- ✓ Lenguajes de propósito general: Java, C, C++, etc.

- ✓ Lenguajes interpretados y procedimentales basados en comandos (scripts): Tcl, Phyton, Perl, etc.

Si no se desea utilizar un lenguaje de propósito general o procedimental basado en scripts, se puede utilizar, como alternativa ideal, un lenguaje especialmente diseñado para el desarrollo de agentes móviles, es decir, que soporte la programación orientada a este tipo de agentes. Estos últimos lenguajes se caracterizan porque son interpretados y orientados a objetos y disponen de todas aquellas funcionalidades y capacidades integradas para dar soporte a las principales características de los agentes móviles como son la autonomía, migración, reactividad, proactividad, seguridad, etc. Un ejemplo de esto último, es el lenguaje Telescript [TELESCRIPT95] similar a Java y que fue el primer lenguaje en ser especialmente diseñado y utilizado para desarrollar el primer sistema de agentes del mismo nombre. Sin embargo, en la actualidad la mayoría de los sistemas de agentes (Aglets, Mole, Odissey, Voyager, Grasshopper, ...) se basan en el lenguaje Java ampliado con nuevas clases para soportar características o atributos de la teoría de agentes móviles.

2.4.2 Lenguajes para la comunicación entre agentes móviles

Al igual que para los lenguajes de desarrollo de agentes móviles, se puede decir lo mismo para el caso de la comunicación entre éstos en los correspondientes entornos de ejecución. Teniendo en cuenta que el modelo de comunicación de los mensajes es por paso de mensajes [PEYNAM03]; en principio, se puede utilizar cualquier lenguaje de programación aunque no esté específicamente diseñado ni para el desarrollo ni la comunicación entre los agentes móviles. Por consiguiente, el programador debe implementar aquellas clases, funciones, procedimientos, comandos, etc., que permita crear, enviar y recibir los datos de los correspondientes mensajes. En este contexto, se destacan los siguientes lenguajes:

- ✓ Lenguajes orientados a objetos: Java ampliado, Telescript, etc.

- ✓ Lenguajes interpretados y procedimentales basados en scripts: Tcl, Phyton, Perl, etc.

En este contexto, conviene reseñar que existen los siguientes tres esquemas para el paso de mensajes entre agentes móviles:

1. Paso síncrono de mensajes (parada y espera):

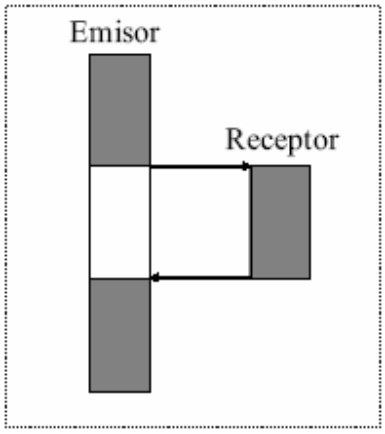

Figura 2.13. Paso síncrono de mensajes.

El emisor bloquea su ejecución cada vez que trasmite un mensaje hasta que el receptor responde al mismo. Este esquema de parada y espera es el más popular y, por tanto, el más usado.

2. Paso asíncrono de mensajes basado en dos envíos:

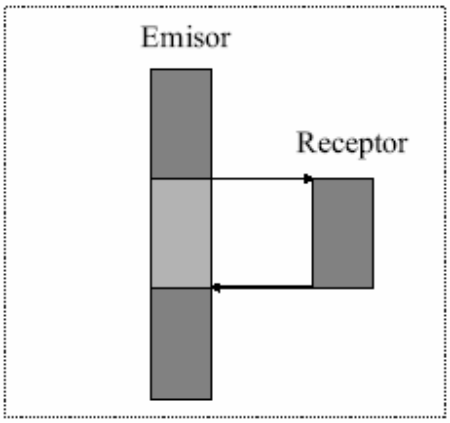

Figura 2.14. Paso asíncrono de mensajes basado en dos envíos.

Ahora, el emisor no tiene que esperar hasta que el receptor responda y envíe la respuesta. Este esquema es más flexible y especialmente útil cuando múltiples agentes interactúan con uno. Para su correcta implementación es necesario mantener en el emisor un identificador del mensaje enviado para no tener que bloquear la ejecución actual e ir preguntando en determinados momentos si hay una respuesta para el mensaje transmitido.

3. Paso asíncrono de mensajes basado en un envío:

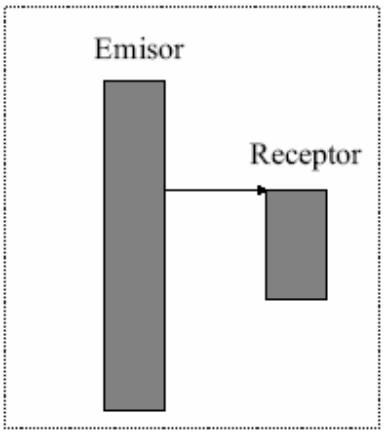

Figura 2.15. Paso asíncrono de mensajes basado en un envío.

Por ser un esquema asíncrono, al igual que en el esquema anterior, no se bloquea la actual ejecución. Ahora, el emisor no retiene el identificador del

mensaje y el receptor no tiene que responder. Obviamente, este esquema es muy útil cuando el agente que transmite un mensaje no espera la pertinente respuesta.

Si no se desea utilizar un lenguaje orientado a objetos o procedimental basado en comandos (scripts), se puede usar, como alternativa ideal, un lenguaje especialmente diseñado para la comunicación entre los agentes móviles. Estos lenguajes se conocen como declarativos o próximos al modelo humano en el sentido de estar basados en sentencias declarativas. Entre dichos lenguajes se destacan los siguientes:

- ✓ KQML (Kowledge Query Management Language) de DARPA [KQLM97]: Sintaxis normalizada de comunicación entre agentes de sistemas heterogéneos.

- ✓ FIPA : Sintaxis similar a KQML [FIPA02].

Se destaca, que estos dos últimos lenguajes están más pensados para la comunicación entre agentes estáticos típicos en el contexto de la Inteligencia Artificial que para la propia comunicación entre agentes móviles.

2.4.3 Plataformas de agentes (Agencias): Clasificación

En este apartado, describiremos brevemente algunos, de los muchos, SA implementados en la actualidad [MAS04]:

- **TELESCRIPT**

  Diseñado por James White (1994) [James94] en la empresa General Magic, Incorporated (California): http://www.genmagic.com. Es considerada la primera plataforma comercial (SA) para el desarrollo y ejecución de agentes móviles. Hay dos conceptos principales en la tecnología TELESCRIPT: *lugares* y *agentes*. Los lugares son las localizaciones virtuales ocupadas por agentes.

Los agentes son los proveedores y consumidores de bienes en las aplicaciones de *mercado electrónico* soportadas por TELESCRIPT.

General Magic ha desarrollado cuatro componentes para soportar la tecnología TELESCRIPT. El primero es el lenguaje TELESCRIPT. Este lenguaje se diseñó para efectuar tareas complejas de comunicación: navegación, transporte, autentificación, control de acceso, etc. (White, 1994). El segundo componente es el motor TELESCRIPT. El motor actúa como un intérprete para el lenguaje TELESCRIPT, mantiene lugares, planifica la ejecución de agentes, dirige la comunicación y el transporte de agentes, y finalmente, provee una interfaz con otras aplicaciones. El tercer componente es el conjunto de protocolos de TELESCRIPT. Estos protocolos tratan de forma primaria con la codificación y decodificación de agentes, para soportar el transporte entre emplazamientos. El componente final es un conjunto de herramientas de software para soportar el desarrollo de aplicaciones TELESCRIPT. Soporta la migración fuerte (ver pag. 21).

En julio de 1995, NTT, AT&T y Sony desarrollaron en Japón un servicio basado en Telescript. En Octubre de 1995, France Telecom anunció una licencia de Telescript para su uso en Francia. Actualmente, General Magic ha suspendido el desarrollo de nuevas versiones de Telescript.

- **AGLETS (AGent+appLETS)**

  Diseñado por Danny B. Lange (1996) [LANGE96] en el laboratorio de investigación de IBM en Tokyo (IBM Tokyo Research Laboratory). Los aglets son agentes autónomos basados en Java, desarrollados por IBM. Proveen las capacidades básicas requeridas para la movilidad. Un aglet refleja el modelo de applet en Java pero brindándole la propiedad de movilidad. Un aglet también puede ser un agente móvil porque soporta las ideas de ejecuciones autónomas y ruteo dinámico sobre sus itinerarios.

  Tiene las siguientes características [IBM Aglets06]:

- Un esquema global único para agentes (Modelo de navegación/seguridad)
- Un itinerario de viaje, para la especificación de patrones complejos de viajes con múltiples destinos y manejos de fallas automáticos (Modelo de navegación)
- Un mecanismo white-board permitiendo que múltiples agentes colaboren y compartan información asíncronamente (Modelo de comunicación)
- Un esquema de transmisión de mensajes que soporta una unión asíncrona desahogada tan bien como una comunicación síncrona entre agentes (Modelo de comunicación)
- Una carga de clases dinámicamente que permite que el código Java de los agentes y la información de su estado viajen a través de la red (Modelo de navegación)
- Un contexto de ejecución que proporciona un ambiente independiente del sistema actual sobre el cual se están ejecutando (Modelo computacional)

Desafortunadamente el modelo de ciclo de vida de un aglet es muy simple y algunas de sus desventajas se mencionan a continuación:

- Soporte inadecuado de control de recursos
- No tiene protección de referencias
- No provee ayuda para la preservación y reanudación del estado de ejecución.

Los aglets utilizan el Protocolo de Transferencia de Agentes independiente de la plataforma para transferir agentes entre redes de computadoras. Soporta migración débil.

TabiCan es un servicio comercial de compra-venta de billetes: http://www.tabican.ne.jp.

- **D'Agents**

  Diseñado por Robert Gray (1996) en la Universidad de Darmouth (Dartmouth College, EE UU): http://www.cs.dartmouth.edu/~agent. Es uno de los primeros SA y nace como propuesta alternativa a Telescript.

  D'Agents [MAS04] es una plataforma simple independiente del sistema de agente móvil. El modelo de navegación esta basado en un simple comando *agente_salta,* este comando puede aparecer en un agente y provocar que su estado y contexto de ejecución sea congelado y transportado a un nodo específico; esta habilidad es más sofisticada que la de los Aglets. El Modelo de Comunicación tiene tres comandos: *agente_enviar, agente_recibir* y *agente_reunir.* Cuando un agente quiere migrar a una nueva maquina, éste manda llamar a una simple función *agente_salta*, el cual automáticamente captura la completa información del estado del agente y la envía al servidor sobre una máquina destino; el servidor destino empieza apropiándose del ambiente de ejecución, cargando la información del estado y retornándolo a su lugar de origen.

  D'Agents tiene importantes características, como:

  - Arquitectura Simple
  - Seguridad
  - Transparencia en la movilidad (TCP/IP)
  - Comunicación entre agentes (RPC)

  El Lenguaje D'Agent es una extensión de Tcl/Tk que soporta la programación distribuida en la forma de agentes transportables. Tcl (*Tool Command Language*) es realmente dos cosas: un lenguaje script y un intérprete para este

lenguaje, que es diseñado para ser fácilmente incorporado dentro de las aplicaciones.

- **ARA (Agents for Remote Action)**

  Diseñado por Holger Peineen de la Universidad de Kaiserlauten (Alemania) en 1997 [HOLGER97], está basado en el lenguaje Tcl/tk (Tool Command Language/tk Toolkit) y dispone de otros tipos de intérpretes incluidos dentro del núcleo de Ara:

  - Permite agentes móviles interpretados en C/C++ basado en MACE (Mobile Agent Code Environment), un entorno interpretativo de ejecución fundamentado en una máquina abstracta.
  - Admite como extensión futura un soporte para Java.

  Dispone de un modelo de seguridad y soporta migración fuerte.

- **MOLE**

  Diseñado por Joachim Baumann (agent control), Fritz Hohl (agent security) and Markus Strasser (fault-tolerant agents) en la Universidad de Stuttgart (Alemania) [MOLE97]. Es la primera implementación de un SA basado en el lenguaje Java (Julio de 1996: Release 1.0 del Sistema de Agentes Móviles Mole). Se caracteriza por la sencillez en su implementación, lo que sirvió para tener pronta acogida en el mundo académico. Posee un entorno gráfico (MOLEview) que permite visualizar la interacción de agentes.

  Cuenta con buenas capacidades de intercambio de mensajes entre agentes, incluyendo mecanismos de seguridad para proteger a los agentes contra máquinas maliciosas. En el primer desarrollo de Mole se usó: RPC y RMI. Soporta migración débil y ejecuciones remotas.

- **ODISSEY**

General Magic Inc. fue el creador del primer sistema de agentes móviles denominado Telescript, cuyo periodo de vida fue muy corto pese a que estaba pensado para trabajar en una arquitectura de red. En respuesta a la popularidad de Internet y al gran auge del lenguaje de programación Java, General Magic decidió reimplementar todos los conceptos considerados en el desarrollo de Telescript pero ahora en este lenguaje. El resultado obtenido fue Odyssey, el cual es una librería de clases Java que permiten al usuario crear sus propias aplicaciones de agentes móviles. Maneja el Protocolo de Transferencia de Agentes Simples (SATP). (General Magic, 2002 http://www.genmagic.com/). Soporta la migración débil.

- **VOYAGER:**

  Desarrollado por la empresa ObjectSpace, Incorporated (Dallas, Texas) en 1998: [VOYAG98]. Está basado en el lenguaje Java e intenta aglutinar las ventajas de “Aglets”, “Mole” y “Odissey”. Soporta movilidad de objetos y agentes móviles para el desarrollo de aplicaciones distribuidas. Dispone de un modelo de seguridad (incluyendo SSL 3.0), soporta la migración fuerte y utiliza como interfaz de transporte: Java RMI, DCOM, CORBA IIOP.

- **CONCORDIA**

  Desarrollado por Mitsubishi Electric Research Laboratories (Cambridge, Massachusetts) en 1998: http://www.meitca.com

  Concordia es un espacio de trabajo para desarrollar y manejar aplicaciones de agentes móviles eficientemente para acceder a información en cualquier tiempo, en cualquier lugar y sobre cualquier dispositivo que soporte Java.

  Las aplicaciones:

  - Procesan datos sobre los datos fuente
  - Procesan datos aún cuando el usuario esta desconectado de la red

- Acceden y entregan la información a través de múltiples redes (LANs, Intranets e Internet)
- Utilizan comunicación inalámbrica
- Soportan múltiples dispositivos clientes, tales como computadoras de escritorio, portátiles, PDAs y teléfonos inteligentes

Concordia oculta las complejidades de programar una aplicación móvil a los programadores, desarrollando una aplicación agente-habilitado similar a un programa estacionario o no móvil. Los agentes mantienen su estado interno mientras viajan en la red, así que ellos pueden reanudar su ejecución al llegar a una nueva posición. Todos los agentes transportan trabajo que se maneja transparentemente sin la intervención del programador.

Un agente Concordia viaja en la red definido por su *Itinerario*. El Itinerario especifica a dónde viajará el agente y qué tareas deberá desarrollar cuando llegue. Los itinerarios de Concordia son especificados en tiempo de ejecución, los agentes pueden cambiar su propio itinerario basados en la información y eventos descubiertos a medida que los agentes viajan.

Algunas de las características que provee Concordia según [NAVA02] son:

- *Servicios de comunicaciones TCP/IP*
- *Manejo avanzado de funciones*
- *Colaboración*
- *Servicio de puentes*
- *Persistencia y manejo de cola*
- *Itinerario*

- *Servicio de nombres*
- *Estructura de seguridad Concordia*
- *Transportador de agentes ligero API*
- *Cifrado*

- **SUMATRA**

  Desarrollado por la Universidad de Maryland en 1996. Basado en el lenguaje Java, soporta la migración fuerte, siendo uno de los pocos sistemas basados en JVM que se han modificado para conseguir una migración transparente de agentes.

2.5 Especificaciones estándar para Sistemas basados en Agentes Móviles

2.5.1 MASIF: Interoperabilidad entre sistemas de agentes móviles

MASIF [MASIF98]: Mobile Agent System Interoperability Facilities es la primera especificación para agentes móviles, la cual fue publicada en 1997 por el grupo OMG para su modelo arquitectónico de referencia OMA/OMG[5]. En concreto, es una facilidad común horizontal accesible vía ORB[6] a través de dos interfaces o componentes: MAFAgentSystem y MAFFinder. Dichos componentes definen operaciones normalizadas con un mismo formato de sintaxis para la gestión y el registro más la localización de agentes móviles, respectivamente.

5 OMA/OMG (Object Management Architecture / Object Managment Group): Arquitectura distribuida y orientada a objetos utilizada en CORBA.

6 ORB (*Object Request Broker)*: Capa de software (también llamada middleware) que permite a los objetos realizar llamadas a métodos situados en máquinas remotas, a través de una red.

- MAFAgentSystem: Define operaciones normalizadas para la gestión básica de agentes móviles [MAS04]:
  - ✓ Crear un agente.
  - ✓ Suspender un agente.
  - ✓ Terminar un agente.
  - ✓ Terminar un Sistema de Agentes (SA).
  - ✓ Obtener el estado de un agente y gestión del mismo (suspender, reiniciar, terminar, etc.).
  - ✓ Obtener información del SA (autoridad, agentes y lugares gestionados, etc.).
  - ✓ Recepción de un agente.
  - ✓ Petición de clases.

- MAFFinder: Define operaciones normalizadas para el registro y localización de los SA, lugares y agentes:
  - ✓ Buscar un agente.
  - ✓ Buscar un SA.
  - ✓ Buscar un lugar.
  - ✓ Inscribir un agente.
  - ✓ Inscribir un SA.

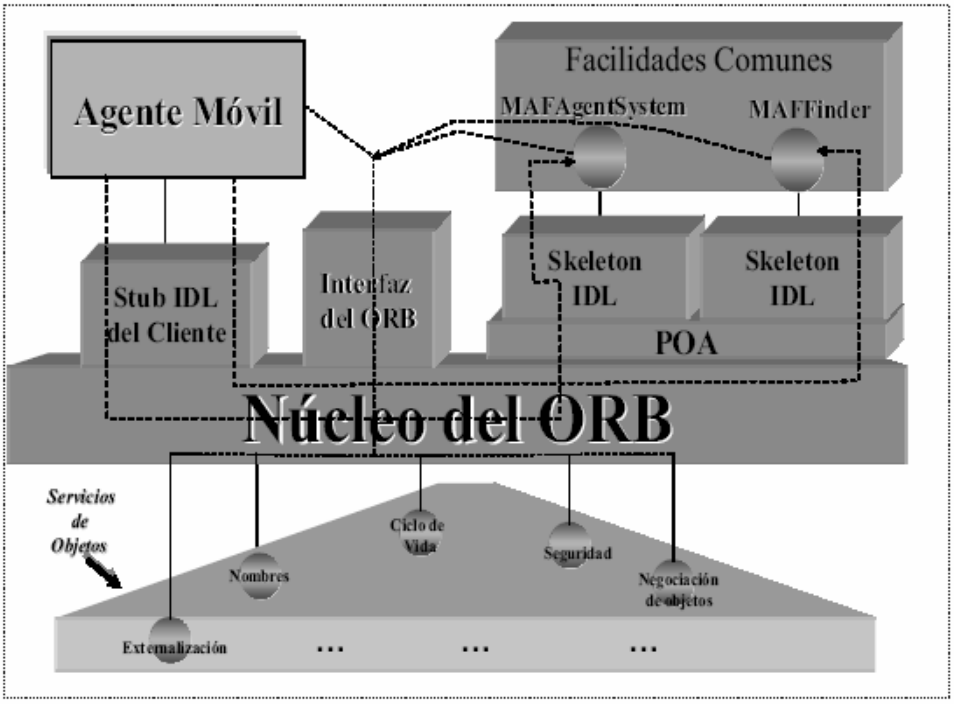

Figura 2.16. Los agentes móviles en la arquitectura OMA/OMG.

La Figura 2.16 muestra cómo se desarrolla un agente móvil en el contexto de la arquitectura de referencia OMA/OMG, o lo que es lo mismo, en el escenario de CORBA. En dicho contexto, un agente móvil se comporta como un objeto CORBA u objeto ORB. Como sabemos, en este escenario todos los objetos o componentes se comunican mediante interfaces IDL o llamadas internas al API de ORB. Asimismo, el bus ORB ofrece un API para que el programador pueda llamar a los métodos de los objetos de la arquitectura OMA/CORBA independientemente del lenguaje empleado.

Según [NAVA02], MASIF se publicó con el objetivo de definir un conjunto de interfaces que permitan la interoperabilidad entre los sistemas de agentes móviles existentes. Al tratarse los agentes móviles como una tecnología emergente, las distintas plataformas existentes en el mercado difieren en arquitectura e implementación, es decir, se consideran sistemas aislados o de carácter propietario carentes de interoperabilidad. Por consiguiente, el objetivo de MASIF es estandarizar algunos aspectos de la teoría de agentes móviles para disponer de una base de interoperabilidad común para dicha tecnología. Asimismo,

conviene destacar que para conseguir la ansiada interoperabilidad entre los SA, éstos deben utilizar el mismo lenguaje de desarrollo para sus agentes. Por tanto, el entorno de ejecución debe estar basado en el mismo lenguaje para poder reanudar o reiniciar la ejecución de un agente en otra plataforma remota. Por ejemplo, sería milagroso retomar la ejecución de un agente en Java en una plataforma para agentes en C.

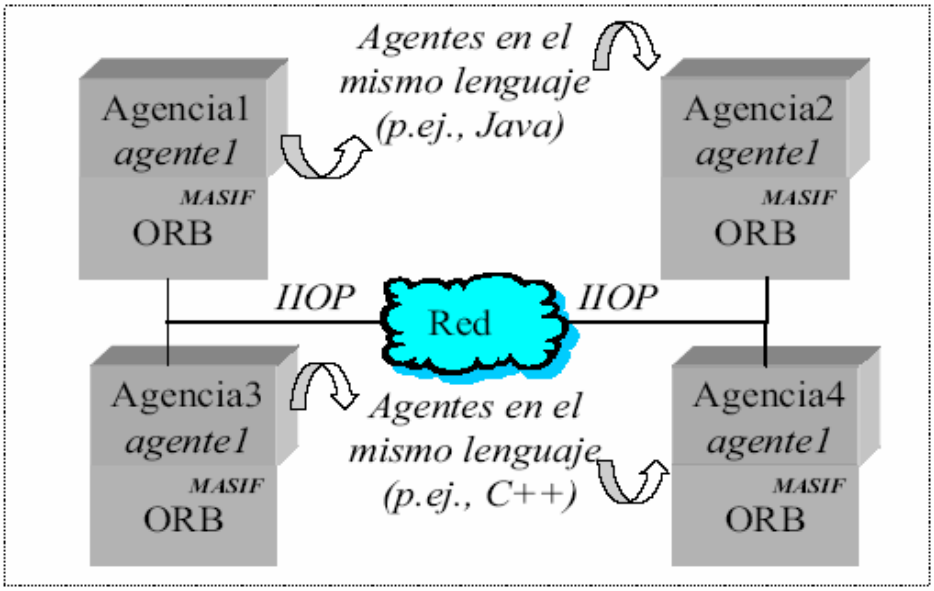

Figura 2.17. Interoperabilidad vía MASIF entre sistemas de agentes móviles.

Los tipos de interacción, entre agentes móviles relacionados con aspectos de interoperabilidad, son los siguientes:

- ✓ Creación remota de un agente:
  - El cliente interactúa con el SA de destino para solicitar la creación de una clase determinada de agente o bien el SA crea el agente de forma proactiva.
  - El cliente se debe autenticar en el SA de destino, estableciendo la autoridad y credenciales que poseerá el agente creado.

- ✓ Transferencia de un agente:

- El SA de origen crea una solicitud de transferencia con la información de nombre y direccionamiento que identifica el lugar de destino.
- Si el SA de destino acepta la transferencia, el SA de origen enviará el estado del agente, su autoridad, sus credenciales de seguridad y, en caso necesario, su código.
- Finalmente, el SA de destino reactivará el agente y continuará su ejecución.

2.5.2 Especificaciones para la Gestión de Agentes FIPA

FIPA (Fundación para los Agentes Físicos e Inteligentes o The Foundation for Intelligent Physical Agents: http://www.fipa.org) [FIPA02] es una organización que comenzó sus actividades en 1995 con el objetivo de estandarizar aspectos relacionados con la tecnología de agentes y sistemas multiagentes. Con esto se pretende facilitar la interconexión entre plataformas de diferentes empresas y organizaciones. El objetivo de FIPA es definir la interoperabilidad entre los sistemas basados en agentes, dejando fuera la implementación interna [MAS04]. Define el modelo de una plataforma basada en agentes, el conjunto de servicios que debe proveer y la interfase entre los servicios.

En la actualidad se puede considerar el estándar más ampliamente reconocido y extendido internacionalmente, y se ha convertido en un referente a seguir para el desarrollo de aplicaciones basados en agentes tanto estáticos como móviles.

FIPA define un entorno de ejecución para agentes llamado plataforma (ver Figura 2.18) que provee sustento al ciclo de vida del agente y las comunicaciones entre agentes, fundamentalmente. Para facilitar el despliegue de una plataforma a través de varias computadoras se definen contenedores como secciones del entorno de ejecución, existiendo siempre un contenedor principal y cero, uno o varios contenedores secundarios.

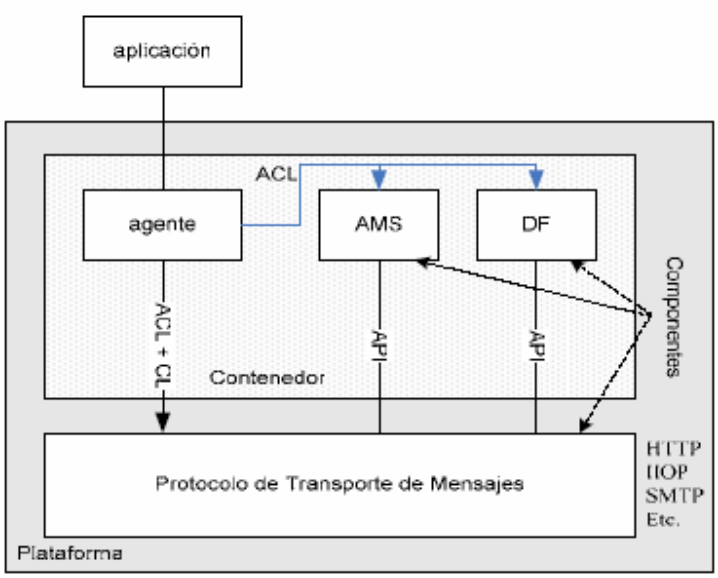

Figura 2.18. Plataforma de agentes de FIPA adoptada por JADE

El contenedor principal aloja tres agentes especializados para la administración de la plataforma:

- ✓ Agent Management System (AMS), que gestiona la plataforma y los agentes, ofreciendo, entre otros, los siguientes servicios:

  - Control de ciclo de vida de un agente, siguiendo su diagrama de estados (ver figura 2.19).
  - Registro de los agentes.
  - Control de movilidad de los agentes.
  - Gestión de recursos compartidos.
  - Gestión del canal de comunicación.
  - Servicio de páginas blancas para localizar agentes por su nombre.

- ✓ Director Facility (DF), que complementa al servicio de nombres, facilitando la localización de agentes según los servicios que ofrecen.

- ✓ Agent Communication Channel (ACC), que gestiona el envío de mensajes entre agentes de una plataforma y entre agentes de distintas plataformas.

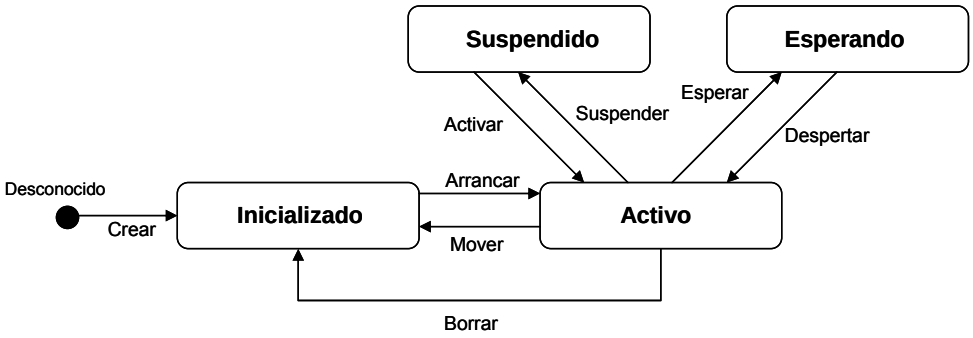

Figura 2.19. Ciclo de estado de los agentes

2.5.2.1 Consideraciones de seguridad en FIPA

En 1998, FIPA realizó la primera especificación para la seguridad [SEGFIPA 98], basando su modelo en el supuesto de que la seguridad debe ser un soporte a la infraestructura y no al agente mismo. El estado actual de ésta especificación es obsoleto, no obstante es de gran utilidad ya que provee algunas ideas para el modelado de una arquitectura de seguridad dentro de una plataforma de agentes basada en FIPA.

El modelo de seguridad ofrecido por FIPA, extiende la funcionalidad de la especificación base (FIPA Agent Management Specification [SEGFIPA 00]), agregando el Agent Platform Security Manager (APSM) a la plataforma, a través del cual pasa toda la información de cualquier otro agente que interactúe con la plataforma. Asimismo, se modificaron algunas funciones del AMS y del DF, y se propusieron extensiones en el metadatos de los mensajes intercambiados entre los agentes para separar niveles de confidencialidad e integridad; es decir, el agente podía configurar el nivel de privacidad e integridad que necesitaba según los requerimientos de la aplicación y las políticas manejadas en ese momento, que podían ser alto, medio o bajo.

La especificación de movilidad de FIPA, considera dos tipos de movilidad: movilidad de dispositivos, que puedan estar conectados de manera intermitente en la red; y movilidad en software, como los agentes móviles los cuales pueden moverse entre nodos en la red. La especificación considera que hay muchas maneras de expresar movilidad dentro de agentes: movilidad del código; migración del agente; y clonación del agente. Asimismo se proponen modificaciones respecto a la especificación original de FIPA en el ciclo de vida de los agentes, los protocolos de comunicación entre agentes y ontologías. Éstas han sido utilizadas en la migración y clonación de los agentes a nivel intra-plataforma. Sin embargo, hasta el momento no hay implementaciones que soporten íntegramente la movilidad de código a nivel inter-plataforma.

2.6 METODOLOGÍAS DE DESARROLLO DE SISTEMAS MULTIAGENTE

El nuevo enfoque del paradigma orientado a Agentes, ha suscitado diferentes aproximaciones, definiciones y metodologías de desarrollo, que se basan en este nuevo modelamiento del “mundo”.

La mayoría de las metodologías orientadas a agentes tienen un sustento teórico basado en metodologías basadas en objetos. Pero a su vez existen algunas que tienen sustentos teóricos de Ingeniería de Conocimiento. A continuación se hará una síntesis de algunas metodologías existentes:

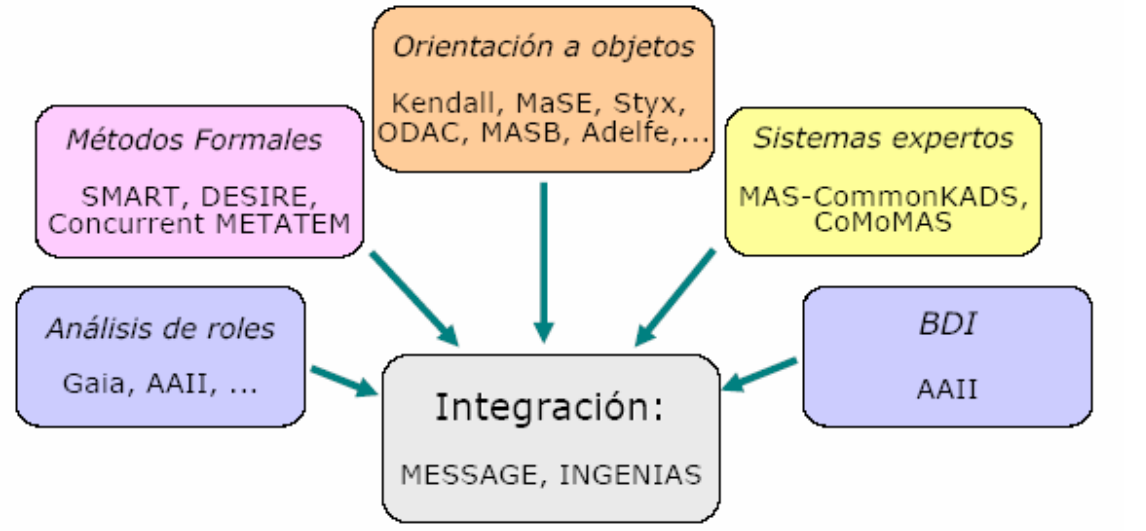

Figura 2.20. Metodologías para desarrollar SMA

2.6.1 Análisis y Diseño Orientado a agentes

Esta técnica, propuesta por Burmeister [BUR96], define tres modelos para el análisis de sistemas basados en agentes: el modelo agente, que contiene a los agentes y su estructura interna (creencias, planes y metas); el modelo organizacional, que describe las relaciones entre los agentes (herencia y roles en la organización); y el modelo de cooperación, que describe las interacciones entre los agentes.

2.6.2 Método basado en escenario para sistemas MultiAgentes (MASB)

Este método (Moulin y Cloutier 1994) es aplicado para sistemas multiagente en el campo de trabajo cooperativo.

La fase de análisis consiste de las siguientes actividades: descripción de escenarios, descripción funcional de roles, modelado conceptual del mundo y datos, y modelo de interacción sistema-usuario.

La fase de diseño consiste de: arquitectura MAS y descripción de escenario, modelado de objetos, modelado de agentes, modelado de conversaciones y validación total del sistema de diseño.

2.6.3 INGENIAS

En INGENIAS [GRASIA05], se persigue un proceso de desarrollo similar al seguido mediante otras herramientas, como Rational XDE, TogetherJ, Enterprise Architect, Paradigm+ que pretenden dar soporte a un proceso iterativo, incremental y basado en componentes (inspirado en RUP [RUP99]). En estas herramientas, el usuario tiene libertad para elaborar diagramas incompletos o generar sus propias vistas del sistema, tomando elementos de diferentes modelos e incorporándolos a un nuevo diagrama. También se intenta separar de UML, en el sentido de no repetir las mismas soluciones. La forma en que se tratan las interacciones en INGENIAS es un buen ejemplo de ello. Las interacciones buscan generalizar diferentes alternativas existentes, como los MSC, diagramas de secuencia o colaboración y diagramas de protocolo basados en AUML (Una extensión libre de UML para el modelado de agentes). Aparte de la generalización, las interacciones se integran completamente dentro de otros modelos como un elemento más. De hecho, la iniciación de interacciones se representa como el producto de ejecutar una tarea.

Otro aspecto tenido en cuenta en INGENIAS es el proceso de generación de los modelos, esto es, qué actividades están involucradas en su producción. El resultado es un conjunto estructurado de actividades detalladas que guiarán a futuros usuarios de la metodología en su utilización. Las actividades se enmarcan en diferentes flujos de trabajo para cada modelo, con lo que se facilita una futura integración de la metodología en entornos automatizados de gestión de proyectos software.

2.6.4 Técnica de Modelado de Agentes para sistemas de agentes BDI

BDI define dos niveles principales (interno y externo) para los agentes BDI (Belief, Desire and Intention).

El punto de vista externo consiste de una descomposición del sistema en agentes y la definición de sus interacciones. Esto se ve a través de dos modelos: el modelo de agente, para describir las relaciones de jerarquías entre los agentes; y el modelo de interacción, para describir las responsabilidades, servicios e interacciones entre los agentes y sistemas externos.

El punto de vista interno se lleva a cabo a través de tres modelos: el modelo de creencias, que describe las creencias sobre el ambiente; el modelo de metas, que describe las metas y eventos que un agente puede adoptar o responder; y el modelo de planes, que describe los planes que un agente puede utilizar para conseguir sus metas.

2.6.5 Metodología MAS-CommonKADS

Esta metodología [CKADS97] comienza con una fase de conceptualización que es una fase informal que se utiliza para recolectar los requerimientos de usuario y se obtiene una primera descripción del sistema desde el punto de vista del usuario.

La metodología define los siguientes modelos:

1. **Modelo Agente:** describe las características principales de los agentes, incluyendo la capacidad de razonamiento, servicios, metas, etc.
2. **Modelo de Tarea:** describe las tareas (metas) que ejecuta un agente y una descomposición de las mismas.
3. **Modelo Expertise:** describe el conocimiento necesario que necesitan los agentes para llevar a cabo las tareas.
4. **Modelo de Coordinación:** describe las conversaciones entre agentes, esto es, sus interacciones, protocolos y capacidades requeridas.
5. **Modelo de Comunicación:** detalla las interacciones humano-agente a través del software, y los factores humanos para desarrollar estas interfaces.

6. **Modelo de Diseño:** recolecta los modelos anteriores y los subdivide en diseño de aplicación, diseño arquitectónico y diseño de plataforma.

2.6.6 Metodología CoMoMAS

Esta es una extensión a la metodología CommonKADS (Schreiber 1994) para el modelado MAS. Se definen los siguientes modelos:

1. **Modelo de Agente:** define la arquitectura de los agentes y el conocimiento de los mismos. Se clasifican como conocimientos sociales, cooperativos, de control, cognitivos y reactivos.
2. **Modelo Expertise:** describe la competencia reactiva y cognoscitiva del agente.
3. **Modelos de Tarea:** describe la descomposición de tareas y detalles si la tarea fue resuelta por un usuario o agente.
4. **Modelo de Cooperación:** describe la cooperación entre agentes.
5. **Modelo de Sistema:** define los aspectos organizacionales de la sociedad del agente conjuntamente con los aspectos arquitectónicos.
6. **Modelos de Diseño:** recoge los modelos previos con el objeto de hacerlos operacionales junto con los requerimientos no funcionales.

2.6.7 Metodología UPSAM

Esta metodología describe el proceso de análisis realizado en el nivel de abstracción de un paradigma orientado a agentes propiamente dicho [SANJUAN06]. Esta metodología basada en el modelo de FIPA y la plataforma JADE, está orientada a la etapa de implementación del sistema de agentes. Se definen los siguientes modelos:

1. **Modelo de tareas.** Este modelo determina los servicios, tareas, objetivos y funcionalidades en general que tiene que ofrecer el sistema de agentes y los asocia a diferentes roles que los agentes habrán de asumir.

2. **Modelo de arquitectura de la organización de agentes.** Este modelo describe como se agrupan los agentes, como son las relaciones entre los agentes, las normas y leyes que tienen que cumplir para asegurar la funcionalidad total del sistema.
3. **Modelo de agentes.** Describe los agentes particulares que van a instanciarse en el sistema y los estados ”mentales” internos que atravesarán a lo largo de su ciclo de vida. Así mismo, describe las jerarquías de agentes del sistema.
4. **Modelo de comunicación de agentes.** Este modelo describe los patrones de comunicación los protocolos de coordinación e intercambio de mensajes entre los agentes internos y externos al sistema.
5. **Modelo de recursos.** Describe los objetos y entidades que pertenecen al sistema, además de los agentes. Describen también el entorno exterior al sistema y cómo es percibido por los agentes del sistema.

2.6.8 Extensiones De UML para el modelado de agentes

2.6.7.1 Aplicación de diagramas de secuencia a Agentes

La figura 2.21 muestra los elementos básicos para la comunicación entre agentes. El rectángulo puede representar tanto agentes de forma individual como también un conjunto de agentes agrupados. Por ejemplo un agente individual puede etiquetarse *Juan/Cliente*. Aquí Juan es una instancia de un agente que está representando el rol de Cliente. Para indicar que Juan es una Persona –independientemente del rol que esté interpretando– Juan puede expresarse como *Juan:Persona*. La manera básica de etiquetar un agente es *nombre-agente/rol:clase*. La sintaxis mencionada antes ya es parte de UML pero con la diferencia que UML indica el nombre de un objeto en lugar de un agente. Se muestra un ejemplo en la Figura 2.

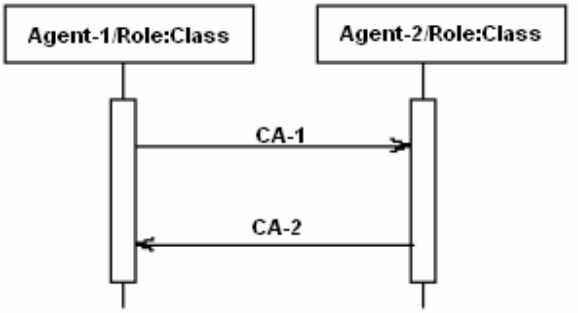

Figura 2.21. Diagrama de Secuencia

Otra recomendación importante de UML es la posibilidad de considerar el soporte para hilos concurrente de interacción. La siguiente figura muestra tres maneras distintas de expresar hilos múltiples. La primera figura indica que todos los hilos CA-1,....,CA-n se envían de manera concurrente.

La segunda figura incluye una caja de decisión que decidirá que CAs (cero o más) se enviarán. Si se envía más de una CA la comunicación es concurrente. La figura 3 muestra una or excluyente, de manera que una sola CA se enviará.

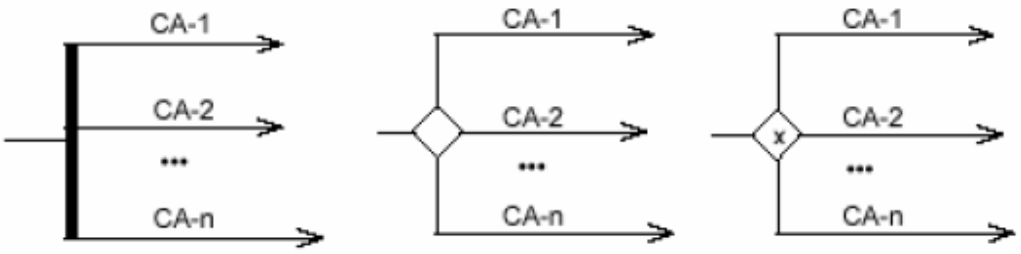

Figura 2.22. Diagrama de Secuencia con OR Excluyente

2.6.7.2 Aplicación de diagramas de colaboración a Agentes

Una de las diferencias existentes con el diagrama de secuencia es que en este diagrama los agentes pueden ser situados en cualquier lugar del gráfico, dando una mayor facilidad al momento de hacer el diagrama. En cuanto al significado del diagrama es el mismo que el diagrama de secuencia, teniendo en cuenta que la

ejecución o llamadas de los mensajes están dados en las etiquetas de los mismos por el número que tienen.

2.6.7.3 Aplicación de diagramas de actividades a Agentes

Los protocolos de comunicación de agentes algunas veces requieren de especificaciones con una semántica muy clara en lo que se refiere al procesamiento de hilos de ejecución. Los diagramas de actividades expresan las operaciones y los eventos que disparan a los hilos de ejecución.

Proveen una representación gráfica que hace posible la visualización de procesos de una manera simple, simplificando de esta manera el diseño y la comunicación del modelo de comportamiento. Por otra parte, se puede representar cualquier tipo de procesamiento concurrente y asincrónico. También pueden representar comunicaciones simultáneas.

3. ESTADO DEL ARTE: SITUACIÓN ACTUAL DEL DESARROLLO DE APLICACIONES MULTIAGENTES PARA DISPOSITIVOS MOVILES

3.1 Importancia de los agentes móviles en entornos inalámbricos

Las características con que cuentan los agentes móviles las hacen muy interesantes para el diseño de aplicaciones en entornos inalámbricos debido a que encapsulan el protocolo de comunicaciones haciendo transparente al programador el tipo de red utilizada. La asincronía de las comunicaciones en la tecnología de agentes móviles nos permite obtener mejores prestaciones en redes inalámbricas. La asincronía de las comunicaciones implica una menor duración de las mismas por lo tanto pueden producirse errores mientras se está procesando la información, lo cual no podría suceder utilizando una aproximación síncrona en las comunicaciones. Además debido a que se reduce el número de reintentos por errores de comunicaciones se reduce el tráfico de red.

Otra característica intrínseca de las redes inalámbricas es que son más lentas que las redes cableadas, por consiguiente si se debe interactuar con un agente o proceso remoto, el agente móvil puede viajar a la máquina en la que se está ejecutando el otro agente/proceso e interactuar localmente más rápidamente que si lo hiciese de forma remota.

3.2 Arquitecturas de agentes para dispositivos móviles

Con la aparición de versiones de Java para pequeños dispositivos, J2ME [J2ME06] para PDA's o teléfonos móviles, y considerando que el lenguaje de programación en el que más plataformas de agentes móviles se han desarrollado es Java, las posibilidades de desarrollar plataformas de agentes en estos dispositivos se presenta alcanzable.

En esta sección se realizará previamente un análisis de las limitaciones, tanto de lenguaje como de recursos, a las que nos enfrentamos a la hora de migrar tecnología de agentes a dispositivos limitados con J2ME y a continuación, se analizarán las propuestas e iniciativas existentes en la literatura relacionadas con este tema.

3.2.1 Tecnología J2ME (Java 2 Micro Edition)

En 1999, Sun Microsystems anuncia la aparición de Java 2 Micro Edition con el propósito de permitir que aplicaciones Java se ejecuten en dispositivos con potencia de procesamiento limitada, como teléfonos móviles, pagers, palm pilots, set-top boxes, y otros. Una solución que responde a la amplia difusión que están teniendo estos dispositivos en los últimos años y a la demanda de usuarios y proveedores de servicios de recibir/ofrecer nuevas aplicaciones para aumentar las funcionalidades que aportan estos pequeños dispositivos.

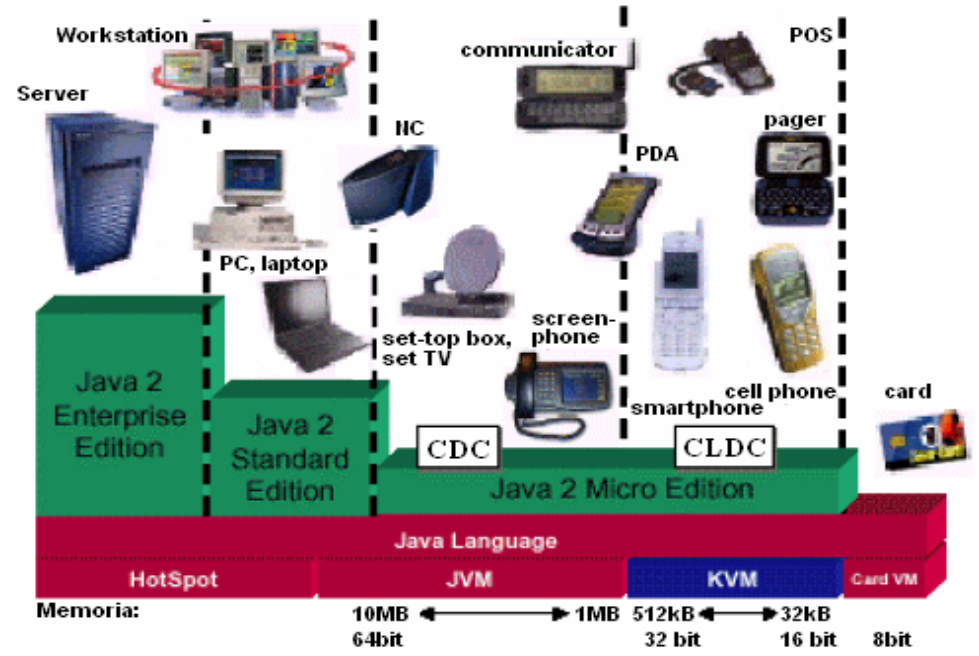

Figura 3.1. Visión por capas de la tecnología Java

J2ME en la actualidad abarca dos categorías de dispositivos, por un parte los que se denominan fijos, que poseen conexiones a la red fija y tienen una capacidad de almacenamiento del orden de los 2 a 16 megaBytes de memoria. Por ejemplo, set-top boxes, Internet TV y sistemas de navegación de automóvil. Y por otra parte, los dispositivos denominados móviles, que tienen capacidades de almacenamiento limitadas del orden de los 128 kiloBytes, con microprocesadores del 16 o 32 bit RISC/CISC y que se comunican a través de conexiones inalámbricas. Dentro de esta categoría están los teléfonos móviles, palm pilots y pagers.

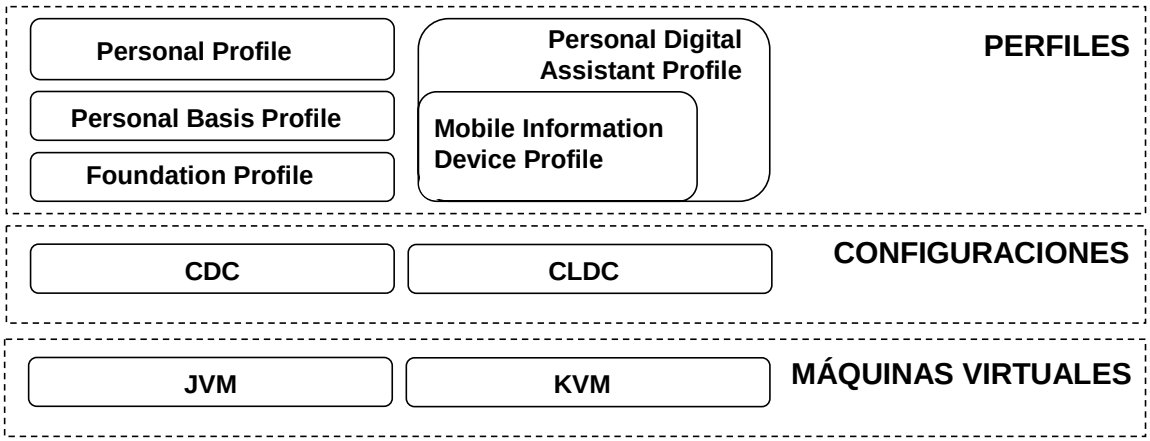

Figura 3.2. Arquitectura J2ME

Una de las principales ventajas de J2ME es que es una arquitectura modular (figura 3.2) que se adapta a las limitaciones de los diferentes dispositivos en las que se quiere integrar. Se definen tres capas:

- ✓ **Máquina virtual.** En la actualidad J2ME soporta dos máquinas virtuales: la Java Virtual Machina que se emplea en ediciones J2SE y en J2EE para los dispositivos con procesadores de 32 bit, y la KVM para arquitecturas de 16/32 bits pero con capacidades de almacenamiento limitado.

- ✓ **Configuraciones.** Definen las características del núcleo de la plataforma. Características soportadas del lenguaje Java, modelo de seguridad y capacidades de interconexión.

  Se establecen una serie de bibliotecas Java que están disponibles para un conjunto de dispositivos, con similares capacidades de procesamiento y memoria. J2ME soporta varias configuraciones, en la actualidad existen dos estandarizadas:

  - ➢ **Connected, Limited Device Configuration (CLDC),** que engloba en general a dispositivos personales móviles.

  - ➢ **Connected Device Configuration (CDC),** que engloba en general a dispositivos fijos. Por motivos de compatibilidad es un superconjunto de CLDC.

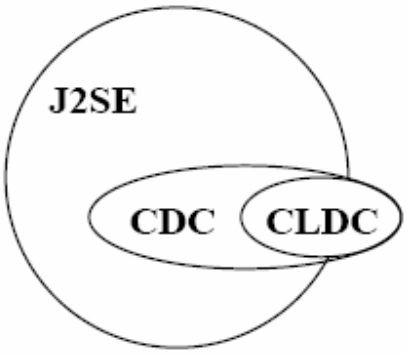

Figura 3.3. Relación entre CLDC y CDC y la J2SE

  Ambas configuraciones tienen clases comunes con la J2SE, que permite la compatibilidad, pero poseen además clases específicas para los tipos de dispositivos para los que se definieron.

- ✓ **Perfiles.** Definen un conjunto de API's que pueden emplearse para desarrollar aplicaciones para una familia particular de dispositivos. El principal objetivo en

la definición de un perfil es garantizar la interoperabilidad de las aplicaciones entre un conjunto de dispositivos que soportan el mismo perfil. Un mismo dispositivo puede soportar diferentes perfiles. Los perfiles se desarrollan sobre una determinada configuración. Así sobre CLDC se ha estandarizado el Mobile Information Device Profile (MIDP) para teléfonos móviles y pagers y se encuentra en proceso de de estandarización el PDA Profile, para asistentes personales. Sobre CDC se están estandarizando el RMI Profile, Foundation Profile, Personal Profile entre otros.

Para el inicio del desarrollo de un sistema de agentes en una plataforma J2ME, es necesario analizar las limitaciones que presentan sus configuraciones. En este sentido, analizamos las limitaciones de los dispositivos que se engloban en la configuración CLDC:

- **Limitaciones del lenguaje Java:**
  - No soporta tipos de datos de coma flotante (float o double), no soporta finalización de instancias de clases y tiene limitaciones en el manejo de errores.

- **Limitaciones de la máquina virtual:**
  - No soporta Java Native Interface (JNI).
  - No soporta cargadores de clase definidos por el usuario.
  - No soporta reflexión, y por lo tanto, ni serialización de objetos, ni soporte a RMI, ni otras características avanzadas de Java (JVM Debugging Interface, JVM Profile Interface)
  - No soporta grupos de threads ni daemon threads, las operaciones de arranque y parada de threads sólo se pueden aplicar individualmente.

La mayoría de estas limitaciones se deben a las propias limitaciones de procesamiento y memoria de los dispositivos y a razones de seguridad motivadas porque J2ME/CLDC no soporta el modelo completo de seguridad de J2SE.

En el futuro se pretenden suplir algunas de estas limitaciones, entre ellas la sincronización de threads y el cargador dinámico de clases.

Si observamos las características que convertían a Java en un buen lenguaje para desarrollar plataformas de agentes, y las comparamos con las restricciones de J2ME, vemos que gran parte de éstas han desaparecido, o se han visto limitadas por motivos de seguridad, en concreto, aquellas que nos permitían implementar movilidad de objetos (carga dinámica de clases, serialización y reflexión).

3.2.2 Tecnología de agentes móviles y J2ME

Existen varias propuestas que pretenden involucrar a dispositivos limitados J2ME en plataformas de agentes móviles, en este apartado haremos referencia a aquellas más importantes.

3.2.2.1 JADE

JADE (Java Agent DEvelopment Framework) [JADE06] es un *middleware* que proporciona tanto un entorno de desarrollo como un entorno de ejecución para la realización y mantenimiento de sistemas multiagente.

El entorno de desarrollo está formado por una serie de librerías en Java que permiten la implementación de agentes de manera limpia e independiente de la plataforma sobre la que se va a ejecutar.

El entorno de ejecución permite a los agentes *vivir* y comunicarse entre ellos. Está realizado enteramente en Java y proporciona una serie de herramientas que permiten al desarrollador controlar y depurar a los agentes en tiempo real.

JADE presenta las siguientes características:

- P2P: Arquitectura peer-to-peer, cada agente puede tomar la iniciativa en una comunicación o bien responder a peticiones que le hagan otros agentes.

- Interoperabilidad: JADE cumple con las especificaciones FIPA, por lo que los agentes desarrollados en JADE pueden interactuar con otros agentes que no tienen porque estar desarrollados con JADE, aunque si deben seguir las especificaciones FIPA.

- Portabilidad: La API que proporciona JADE es independientemente de la red sobre la que va a operar, así como de la versión de Java utilizada, teniendo la misma API para J2EE, J2SE y J2ME.

- Intuitiva: JADE se ha desarrollado para ofrecer una API fácil de aprender y sencilla de manejar.

Los agentes JADE tienen nombres únicos y se permite a cada agente descubrir a otros agentes y comunicarse con ellos mediante comunicaciones punto a punto. Los agentes proporcionan servicios, cada agente puede buscar a otros dependiendo de los servicios que proporcionen otros agentes.

La comunicación entre agentes se lleva a cabo a través de mensajes asíncronos, es decir, el agente que envía el mensaje y el destinatario del mensaje no tienen porqué estar disponibles al mismo tiempo. Es más, el destinatario no tiene porqué existir en ese instante. Los mensajes se pueden enviar a un agente en concreto o se pueden enviar a agentes que se desconocen pero se sabe que poseen unas ciertas características. JADE proporciona mecanismos de seguridad, ya que habrá agentes a los que no se les esté permitido comunicarse con otros agentes, de manera que una aplicación puede verificar la identidad del receptor y del agente que envía el mensaje y no dejar realizar actuaciones no permitidas para un

determinado agente. La estructura de los mensajes se basa en el lenguaje ACL (Agent Communication Lenguaje) que ha sido definido por la FIPA.

JADE también permite a los agentes cambiar de Host (en J2SE). Un agente puede interrumpir en un momento dado su ejecución, migrar a otro host (sin necesidad de que el código esté previamente en el host destino) y continuar la ejecución en el mismo punto en el que la interrumpió. Esto permite un balanceo de carga ya que permite a los agentes migrar hosts menos cargados.

El entorno de ejecución proporciona un marco donde poder ejecutar los agentes y herramientas gráficas para su monitorización y depuración.

3.2.2.1.1 Arquitectura

Los agentes JADE necesitan del entorno de ejecución donde poder *"vivir"*. Cada instancia del entorno de ejecución se denomina *contenedor* (container). Al conjunto de los contenedores se le denomina *plataforma* (platform) y proporciona una capa que oculta a los agentes (y al desarrollador) el entorno donde se ha decidido ejecutar la aplicación [PRJADE02].

En cada plataforma debe existir un contenedor especial denominado *contenedor principal* (main container). La principal diferencia del contenedor principal respecto al resto de contenedores es que alberga dos agentes especiales:

- **AMS** (Agent Management System): Este agente proporciona el servicio de nombres asegurando que cada agente en la plataforma disponga de un nombre único. También representa la autoridad, es posible crear y matar agentes en contenedores remotos requeriendoselo al agente AMS.

- **DF** (Directory Facilitator): Proporciona el servicio de *Páginas Amarillas*. Gracias al agente DF, un agente puede encontrar otros agentes que provean los servicios necesarios para lograr sus objetivos.

La arquitectura se puede observar en la figura 3.4, donde aparecen dos plataformas diferentes (platform1 y platform2), cada una con sus contenedores principales y contenedores normales.

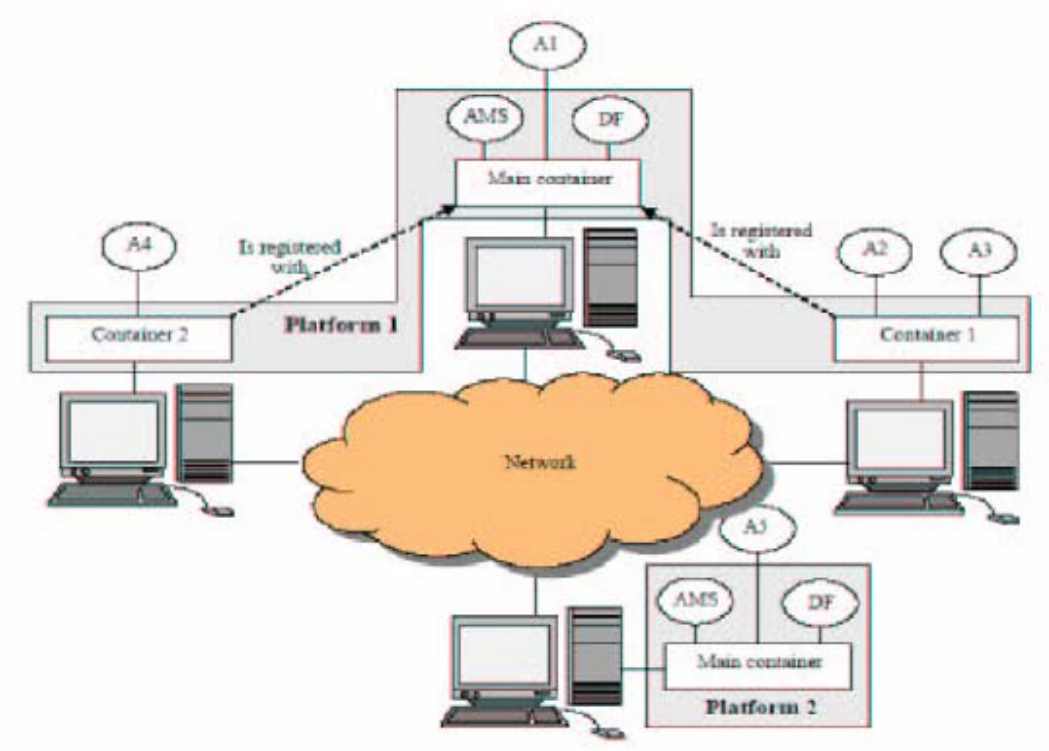

Figura 3.4: Esquema de distribución de los containers y las plataformas.

3.2.2.1.2 LEAP

Los agentes escritos en JADE pueden ejecutarse en entornos móviles como teléfonos o PDAs integrando las redes inalámbricas junto con las redes convencionales. El problema es que JADE no puede correr en pequeños dispositivos debido a diversas razones:

- **Espacio:** La memoria necesaria para el entorno de ejecución es de varios megas.

- **Version del JDK:** JADE necesita el JDK 1.4 o posterior, mientras que la mayoría de los dispositivos solo soportan PJava (Personal Java) o bien MIDP.

- **Características propias de las redes inalámbricas:** Como son IP dinámicas, conectividad intermitente o bajo ancho de banda, hace que JADE no sea lo apropiado.

Para ello surge LEAP (Lightweight Extensible Agent Platform) [LEAP06] que permite ejecutar agentes JADE en dispositivos móviles y/o conectados a través de redes inalámbricas.

El proyecto LEAP engloba un consorcio de compañías entre las que se encuentran, entre otras, Motorola, British Telecom y Siemens. El objetivo del proyecto es el desarrollo de una plataforma de agentes en Java conforme al estándar FIPA (Foundation for Intelligent Physical Agents) que pueda operar tanto en dispositivos limitados (teléfonos móviles, PDA, pagers) con J2ME, como en PC's con J2SE. Para ello, han diseñado una arquitectura modular, con una parte obligatoria, común a todos los tipos de dispositivos, y otra opcional; y mediante un instalador se podría componer la plataforma según las limitaciones del dispositivo en el que se instale.

Los container del entorno de ejecución de JADE-LEAP se dividen en dos partes, un FrontEnd, que se ejecuta en el dispositivo móvil, y un BackEnd, que se ejecuta en un servidor de la red fija, como se muestra en la figura 3.5

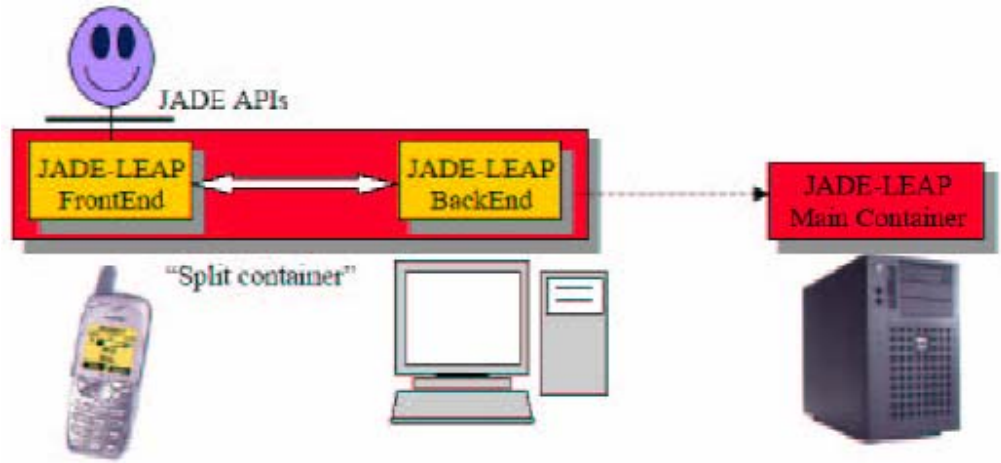

Figura 3.5. Ejecución de dos agentes en dispositivos móviles y su arquitectura vista en el RMA.

Este tipo de arquitectura ofrece una serie de ventajas:

- El FrontEnd necesitará los recursos mínimos.
- La IP del dispositivo móvil va a permanecer oculta a otros containers.
- Se va a permitir un control en la conexión entre el BackEnd y el FrontEnd, de manera que si se produce una desconexión, entre el dispositivo móvil y la red, no se va a producir una perdida de mensajes.
- La cantidad de bytes transmitidos en la red inalámbrica es menor.
- El tiempo de carga del programa es menor.

Aunque como contrapartida, no podremos crear agentes móviles. Si ejecutamos el ejemplo que nos viene con el entorno JADE-LEAP (necesitamos tener el kit de desarrollo Wireless Toolkit de Java, ya que proporciona un emulador de dispositivos móviles) en el emulador proporcionado por el WTK (Wireless Toolkit) podremos ver en la interfaz gráfica (figura 3.7) del RMA los containers creados, los

FrontEnd y los BackEnds lanzados. El ejemplo proporcionado es una aplicación chat, en el que las personas pueden chatear con sus teléfonos móviles.

Figura 3.6: Emuladores donde están corriendo agentes JADE.

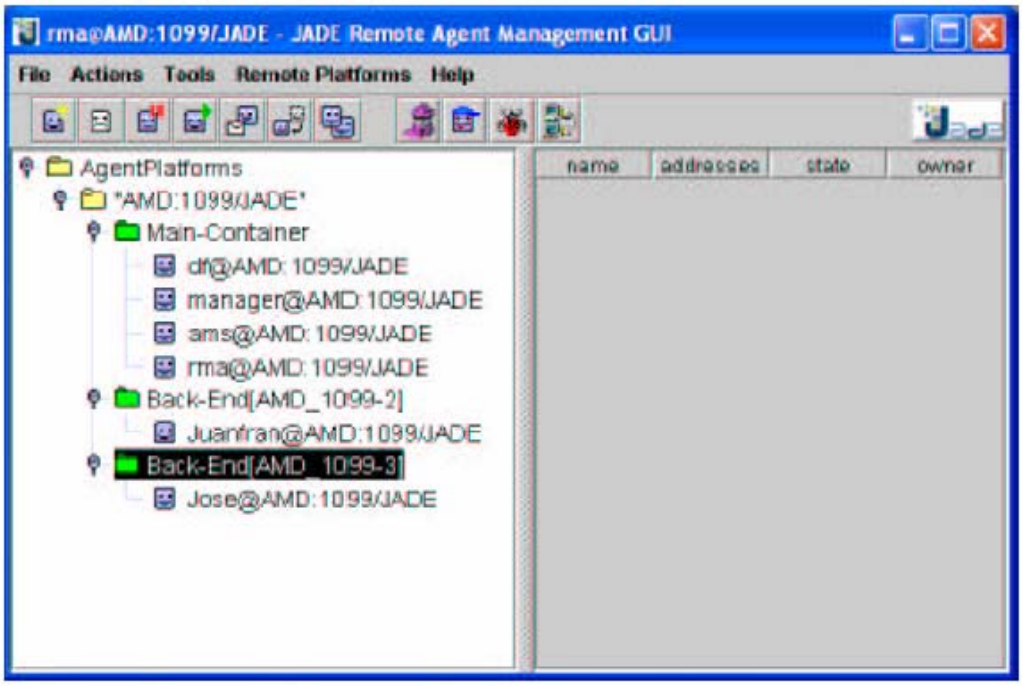

Figura 3.7: Ejecución de dos agentes en dispositivos móviles y su arquitectura

3.2.2.2 Monash University

En Monash University [MONASH00], [MONASHEC01] se están realizando proyectos relacionados con el desarrollo de plataformas de agentes para dispositivos móviles empleando J2ME, en particular se han realizado desarrollos para PDA. Su propuesta se basa en incluir dentro de la KVM una plataforma de agentes que habían desarrollado para entorno PC, con algunas restricciones, obteniendo de esta forma unas mejores prestaciones, pero obligando a que los dispositivos tengan esta KVM reconstruida.

3.2.2.3 School of Computer Science of Carleton University

En School of Computer Science of Carleton University [CARLETON01] también se están realizando desarrollos orientados a la utilización de tecnología de agentes para aplicaciones en entornos móviles, pero en su propuesta la plataforma de agentes reside en un dispositivo no limitado denominado Agent Gateway que sirve de mediador entre el dispositivo inalámbrico y los recursos de la red. La justificación de esta propuesta es que los dispositivos móviles tienen recursos limitados y que en la actualidad la especificación de la J2ME/CLDC no tiene funcionalidades básicas para la realización de plataformas de agentes, como es la serialización de objetos o la carga dinámica de clases.

4. ESTADO DEL ARTE: SITUACIÓN ACTUAL DEL DESARROLLO DE JUEGOS PARA DISPOSITIVOS MÓVILES

Antes de comenzar a estudiar y evaluar el desarrollo de un juego para un terminal móvil, es de vital importancia conocer tanto qué tipos de juegos existen, como las limitaciones de la plataforma en la que estamos desarrollando. Por ello, esta sección pretende primero hacer una clasificación de los juegos que han ido apareciendo desde los primeros Space Invaders y Pac-Man hasta nuestros días [RUSEL02], y sobre todo ver qué tipos de juegos son más apropiados para un terminal móvil, y cuáles serán inviables. Para finalizar, intentaremos ver cómo afrontar un proyecto real desde el punto de vista de Director de Proyectos, hablando del mercado actual de entretenimiento y estimando el equipo humano que debería llevar a cabo un proyecto de esta índole con éxito.

4.1 Tipos de Juegos

En 1962, Steve Russell, un investigador del MIT (Massachusetts Institute of Technology), diseñó un juego llamado ***Space War*** [SPACEWAR06], el primer juego diseñado para un ordenador con pantalla. Pero no fue hasta comienzos de los años 70 que se empiezan a comercializar los primeros vídeojuegos, *Computer Space* (1971) y *Pong* (1972) ambos desarrollados por Nolan Bushnell, fundador de Atari. Éstos que ahora nos parecen triviales, eran entonces un alarde de tecnología y en seguida se hicieron populares en todo el mundo.

4.1.1 Arcades

Los primeros juegos fueron los llamados de tipo ***Arcade***. Se caracterizan por ser juegos muy simples que ponen a prueba nuestros reflejos y nuestra habilidad, muy rápidos, cuya dificultad radica en una velocidad creciente según avanzamos en el

tiempo, o superamos niveles. Los gráficos suelen ser muy esquemáticos y las pantallas suelen ser muy parecidas entre sí, cambiando sólo la disposición de obstáculos y enemigos. Esos obstáculos se suelen llamar *Tiles* (tejas). En estos juegos no existe por tanto *Scroll* (explicado más adelante). Ejemplos de este tipo de juegos son: *Space Invaders* (1978) y *Pac-Man* (1983), así como el *Ping-Pong* y su evolución posterior *Arkanoid.* Juegos ya portados a teléfonos móviles.

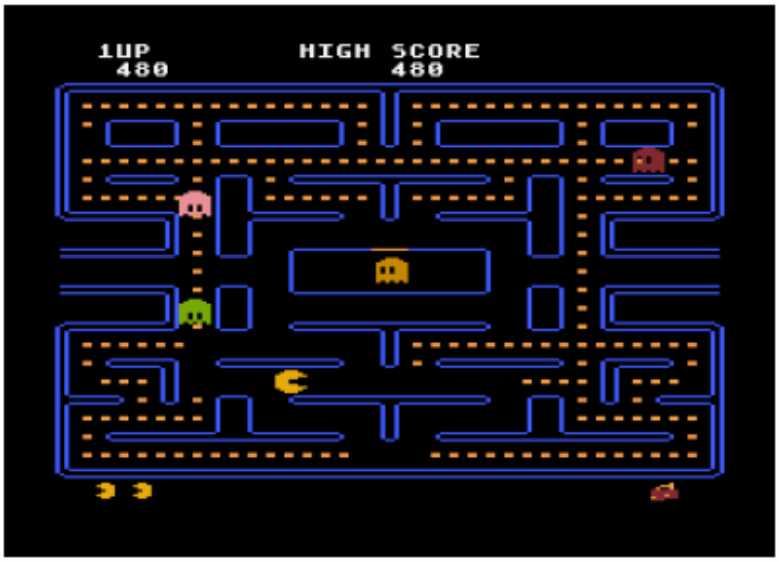

Figura 4. 1. Pac-Man (1983)

Estos juegos por su sencillez, son rápidos de desarrollar (y por tanto baratos), no demandan apenas recursos. Sin embargo, no hay que olvidar que este tipo de juegos engancharon a varias generaciones de los años 70 y 80.

4.1.2 Shoot'em up

Otro tipo de juegos tremendamente populares son los llamados ***Shoot`em up*** (Acaba con ellos). Se pueden considerar como un subtipo de los Arcades. Son juegos de acción en los que aparecerán multitud de oleadas de enemigos (Enemy Waves), a los que tendremos que aniquilar mientras avanzamos bien de forma horizontal, bien de forma vertical por un escenario. Este desplazamiento es lo que técnicamente se llama *Scroll*. Con el tiempo y la mejora de la tecnología se

crearon juegos más realistas utilizando una técnica llamada *MultiScroll.* Consistente en que hay varios planos de *Scroll* moviéndose más lentamente cuanto más alejado de la cámara se encuentre. Gracias a esta técnica se conseguía dotar de profundidad a la acción, en momentos en que no se podía ni plantear la posibilidad de efectos 3D.

Ejemplos de este tipo de juegos son: *R-Type,* el *Nemesis* y más tarde la saga *Siberian Strike* que ya ha sido portada a los móviles, aunque posteriormente se abandonó el recurrente tema de matar alienígenas y aparecieron títulos míticos como el *Commando* ambientado en Vietnam, el cuál fue imitado hasta la saciedad incluso en España con el *Desperado*. Juego de la época del Spectrum ambientado en el Oeste americano.

Figura 4. 2. R-Type

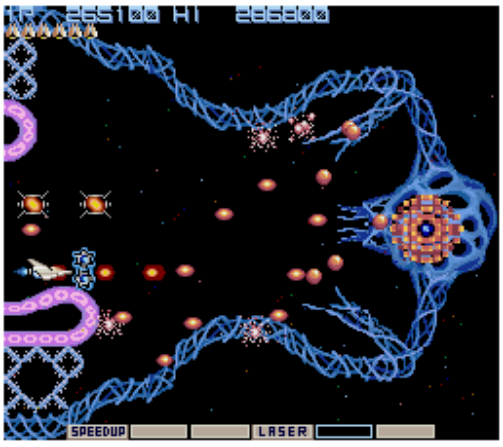

Figura 4.3. Nemesis

Figura 4.4. Siberian Strike Java MIDP

Figura 4.5. Siberian Strike para teléfonos de gama baja

4.1.3 Plataformas

En Septiembre de 1985, Nintendo sacó al mercado en Japón el juego de ***Plataformas*** por excelencia, el *Mario Bros*. Estos juegos se caracterizan por una visión lateral de la escena, como si viéramos un corte transversal de un edificio. En los primeros juegos de plataformas como el *Donkey Kong* (1983), la acción transcurría en una pantalla estática, pero en cuanto los recursos de las máquinas lo permitieron, estos juegos fueron dotados de *scroll* bien horizontal, bien vertical. La dinámica de estos juegos consiste básicamente en avanzar en alguna de las cuatro direcciones posibles, saltando de plataforma en plataforma, esquivando enemigos y obstáculos, recogiendo objetos que proporcionen puntos, y llegar al final del nivel antes de un tiempo determinado.

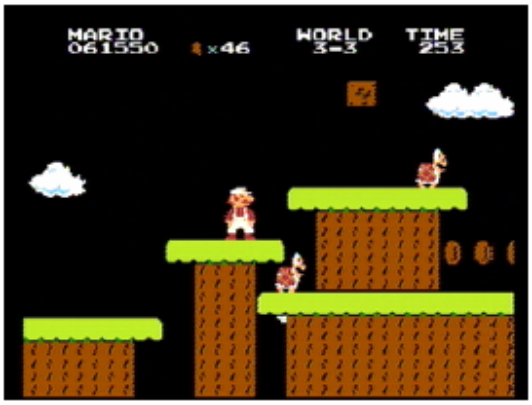

Figura 4.6. Mario Bros (1985)

Figura 4.7. Donkey Kong (1983)

Este tipo de juegos son perfectos para ser portados a los teléfonos móviles. Cumplen todos los requisitos deseables en cuanto a sencillez de los gráficos, y manejo. Y es en el manejo donde tenemos la mayor de sus ventajas. Para jugar no necesitamos más que una tecla para cada una de las cuatro direcciones y un par de teclas adicionales para saltar y disparar. Y es que casi todos los móviles actuales, no tienen teclas para las cuatro diagonales. Lo cuál no es un problema para este tipo de juegos.

Son juegos ligados a las máquinas recreativas y las consolas de vídeojuegos, como por ejemplo el *Bubble Bobble* y el *Sonic*, aunque también llegaron a los PCs (*Rick Dangerous*, *Rayman 3*...).

Figura 4.8. Bubble Bobble

Figura 4.9. Rick Dangerous II

4.1.4 Aventuras Conversacionales

En una época en la que todavía se tenía que utilizar la imaginación para sumergirse en la atmósfera que intentaban recrear los juegos, tuvieron bastante éxito las llamadas **Aventuras Conversacionales** (**Interactive Fiction**). Inicialmente consistieron en historias textuales donde se describía al jugador la situación en la que se encontraba, y se le permitía realizar algunas acciones sencillas , como moverse en las direcciones de los cuatro puntos cardinales, recoger objetos e incluso hablar con personajes, pero siempre escribiendo unos estrictos comandos textuales. Más tarde este tipo de juegos incorporaron gráficos,

de tal modo que al movernos a otra dirección veíamos una nueva imagen, que era fundamental para completar la aventura ya que nos daba pistas visuales. Sin embargo nunca superaron el engorroso interfaz textual en el que muchas veces teníamos que acertar con la frase exacta para que el programa entendiera la acción que queríamos realizar.

Este tipo de juegos serán extremadamente sencillos de programar. Lo fundamental será dar con una buena historia que enganche al jugador. Por los escasos recursos necesarios, éstos juegos podrían ejecutarse en el terminal más modesto que soporte Java, sin embargo deberíamos encontrar alguna buena manera de simplificar el interfaz textual, pues nadie jugará a nuestro juego si tiene que escribir interminables frases para interactuar con él. Desarrollar aventuras conversacionales en el siglo XXI puede parecer ridículo, pero si pensamos en la cantidad de gente de las grandes ciudades que llevan su libro todos los días en el transporte público, puede que tengamos miles de potenciales usuarios, a los que les podremos vender libros con muchos valores añadidos, y que llevarán siempre disponible en su móvil. Un ejemplo cercano de este tipo de juegos es el juego español *Don Quijote de la Mancha*, muy popular en la época del Spectrum.

Figura 4.10. Don Quijote de la Mancha

4.1.5 Perspectiva Isométrica

En otro intento por incorporar la tercera dimensión en los vídeojuegos, apareció un nuevo concepto, la **Perspectiva Isométrica**. Hasta ese momento todos los juegos usaban una vista cenital o lateral, ya que era impensable generar gráficos convincentes en 3D. Se pensó entonces en crear todos los gráficos del juego en perspectiva, vistos desde varios ángulos distintos, pero tratarlos como si fueran imágenes planas. De este modo, toda la complejidad residía a la hora de tratar con las coordenadas, que eran interpretadas en ese plano isométrico. Se conseguían gráficos muy detallados con pocos recursos de la máquina, aunque el manejo del personaje era más complicado, y los jugadores tuvieron que acostumbrarse a una nueva manera de explorar mundos virtuales. Precisamente por esa dificultad de manejo, este tipo de juegos no requerían de respuestas rápidas sino de resolver puzzles y enigmas, de dar saltos precisos para alcanzar nuevos niveles, usar objetos y esquivar a enemigos que normalmente realizaban movimientos cíclicos.

Este tipo de juegos es también factible en terminales móviles. Son bastante más complejos de desarrollar que los comentados anteriormente, ya que además de las características señaladas, será imprescindible para un juego de este tipo, guardar la partida para que el usuario pueda recuperarla justo en el punto donde la dejó.

Ejemplos de este tipo de juegos son el *Head over Heels*, *Batman*, ambos de la época de los ordenadores de 8 bits y más recientemente, *Looney Tunes* para Gameboy. Éstos juegos eran tan divertidos que aún hoy se puede encontrar en Internet gente que ha programado un remake para poder seguir disfrutando de ellos en PC.

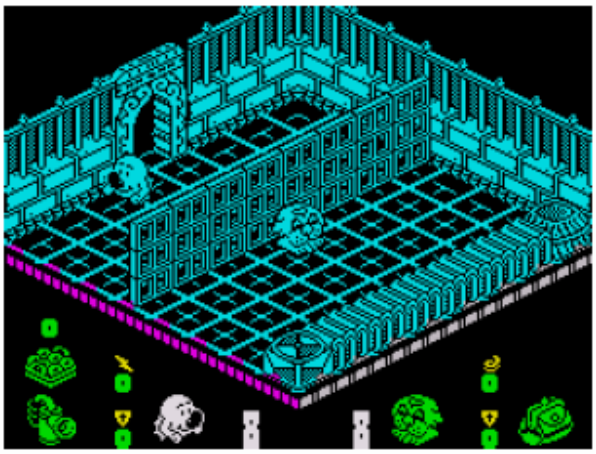

Figura 4.11. Head over Heels (1987)

Figura 4.12. Head over Heels Remake

4.1.6 Estrategia

Un género que cuenta con una cantidad ingente de adeptos es el de los juegos de ***Estrategia o de Gestión de recursos***. Éstos se clasifican en dos tipos, ***Por Turnos*** y de ***Tiempo Real***. Ambos se diferencian en cómo transcurre la acción. En los primeros se le da a cada jugador un tiempo, que puede ser infinito. El jugador toma decisiones, realiza sus movimientos y cuando ha terminado pasa turno, para que el siguiente jugador (que puede ser manejado por la máquina) haga sus movimientos. En los de tiempo real, todos los jugadores están realizando acciones consecutivamente, por lo que no sólo será importante tener una buena estrategia sino también llevarla a cabo cuanto antes. Al tratarse de varios

jugadores simultáneos, son juegos que deberán ser jugados en red, o bien un único jugador contra la máquina, ya que deben ser jugados con ratón, y sólo puede haber uno activo por ordenador.

En cuanto al tema que nos ocupa realmente, que es el de si este tipo de juegos puede ser apto para los móviles, decir que deberemos descartar los de tiempo real sin la ayuda de la tecnología de agentes móviles. Por otro lado, los controles habituales de un móvil harían que el usuario se desesperara ante la impotencia de poder indicar acciones rápidas. Sin embargo, ya existen en el mercado terminales como el Sony-Ericsson P990i (http://www.sonyericsson.com), que incorporan una pantalla táctil y lápiz, que harían posible jugar de una forma ágil. Además, desde el punto de vista de la programación, desde la primera versión de JAVA MIDP, ya tenemos métodos implementados para lidiar con este interfaz, típico de las agendas electrónicas. No debemos olvidar sin embargo, el otro tipo de juegos comentado anteriormente. Una estrategia por turnos es susceptible de ser jugada en un terminal móvil porque no importa cuánto tardemos en acabar nuestro turno, además la capacidad on-line de éstos se puede utilizar para permitir partidas multijugador. Esto que sería descabellado en la mayoría de los juegos, no lo es en absoluto en este caso, pues la información que se debería intercambiar al final de cada turno sería realmente pequeña, como, la posición en la que quedaron las tropas, energía, munición, los recursos que ha ganado/perdido en el turno, etc. información que sería de pocos bytes, y por tanto muy apropiada para redes GPRS donde se factura por byte enviado y no por tiempo, resultando asequible para el usuario.

Ejemplos de juegos de estrategia en tiempo real son el *Command&Conquer* y el *War Craft*. De gestión de recursos está el *Sym City* o el *Train Tycoon.* De estrategia por turnos tenemos la saga de *Heroes of the Migth and Magic*

Figura 4.13. Heroes of might and Magic

4.1.7 Rol

Aunque tienen muchas similitudes con el tipo de juegos comentado en el apartado anterior, los ***Juegos de Rol*** (Role Games) deben ser tratados aparte. Los juegos de rol son muy anteriores a la era digital. Y hasta que dieron el salto a los ordenadores, era algo parecido a quedar con los amigos para jugar a un juego de mesa y pasar una tarde entretenida. Sin embargo los juegos de rol son mucho más que eso, para empezar no suele haber ni siquiera tablero, simplemente se necesita lápiz, papel y dados. Uno de los participantes es el llamado *Máster*, y es el encargado de llevar preparada una historia en la que mediante una narración descriptiva, inmiscuirá a los demás jugadores en ella, indicándoles lo que sienten y ocurre en cada momento. Cada jugador asumirá un papel (role) en el juego, y deberá comportarse como tal. Cada jugador deberá rellenar una ficha con las características de su personaje, objetos que lleva, puntos de vida, etc. y que se irán actualizando en el transcurso de la historia. El factor aleatorio lo ponen los dados. Todo el tema de tirar los dados, calcular y apuntar daños recibidos, etc., le restaba dinamismo al juego, y hacían que una lucha pudiera ser interminable y difícil de llevar, eso quizás fue lo que llevó a estos juegos a los ordenadores. Todos estos cálculos se hacían de forma automática, pero los gráficos que podían mostrar los ordenadores de la época, nada tenían que hacer con la gran imaginación de los jugadores habituales de Rol. Fue por este hecho que al

principio no se hacía especial hincapié en los gráficos, sin embargo con el tiempo, los gráficos empezaron a tomar más relevancia, hasta tal punto que fueron pioneros en el uso de perspectiva en primera persona en 3D, como por ejemplo el *Drakkhen*, y un tiempo después el *Dungeons&Dragons III.*

Este tipo de juegos no serán apropiados para los móviles por la misma razón que los de tiempo real, sin embargo sí serían viables si se utiliza un enfoque por turnos, cosa poco común en estos juegos que resultan a veces difíciles de manejar incluso con ratón.

Un ejemplo puramente considerado juego de Rol es el *Icewind Dale*.

Figura 4.14. Ice Wind Dale

4.1.8 Lucha

Un género revolucionario fue sin duda el de los ***Juegos de Lucha***. Empezaron siendo una evolución de los *Shoot'em up* en los que a parte de disparar, se podían realizar distintos tipos de golpes, hasta convertirse en un nuevo género en sí mismos. El juego mítico que creó escuela fue el *Double Dragon*, un claro ejemplo de esta evolución de los *Shoot'em up*. Más adelante CAPCOM sacó el *Street Fighter*, juego que se centraba exclusivamente en la lucha, en el que el usuario debía realizar movimientos complejos y sincronizados junto con la pulsación de

uno o varios botones, para conseguir golpear al adversario. Cuanto más compleja fuera la secuencia, más efectivo sería el golpe. Este juego se convirtió en un auténtico fenómeno de masas y abarrotó los ahora vacíos salones de recreativos a finales de los años 90, y provocó la aparición de numerosos clones y versiones (*Mortal Kombat*).

En cuanto a si esta variedad de juegos es o no apropiada para la plataforma que nos ocupa en este trabajo, diríamos que no si no fuera porque recientemente ha aparecido la versión oficial de *Street Fighter* para móviles. La razón obvia por la que no parece recomendable, es la falta de un Joystick. Y es que no hay que olvidar que la "gracia" de los juegos de lucha reside en la destreza de los jugadores para realizar movimientos rápidos, precisos y sincronizados antes que su adversario. Con las teclas de un móvil resultaría realmente trivial realizar golpes. Además, técnicamente estaríamos limitados para la detección mediante JAVA, de la pulsación simultánea de un número elevado de botones. Sin embargo, sí puede ser factible, la implementación de un *Shoot'em up* que permitan un número muy limitado de golpes sencillos de realizar, siguiendo el estilo creado por *Double Dragon* y *Golden Axe.* Además sería muy interesante permitir dos jugadores simultáneos mediante Bluetooth (viable sólo en terminales compatibles con MIDP 2.0).

Figura 4.15. Street Fighter 2

4.1.9 Aventuras Gráficas:

Las ***Aventuras Gráficas*** son un género de juegos con unas características muy bien definidas. Fueron un nuevo concepto de juego que aparecieron gracias a la popularidad del PC. Hasta entonces, los juegos que existían estaban pensados para las máquinas recreativas, donde lo que interesaba es que las partidas fueran cortas para recaudar más. Sin embargo, las Aventuras Gráficas son juegos para ser jugados durante semanas o incluso meses. No sólo eso resultó novedoso, sino también la forma de interactuar con el juego, en el que se utilizaba masivamente un periférico nuevo en ese momento, y que no tenían las tragaperras, el ratón. La mecánica del juego consiste en pantallas estáticas, (gracias a lo cuál se pueden utilizar gráficos bastante complejos), normalmente consistentes en fondos creados por un dibujante donde existen pequeñas animaciones de objetos, a parte de las de los personajes del juego. El jugador pincha sobre cualquier área de la imagen estática, y el protagonista se mueve hasta ese punto. La imagen de fondo aunque estática, suele tener perspectiva, de modo que mediante efectos muy sencillos, nos dará la impresión de que nuestro personaje se aleja y se acerca a la cámara según la parte de la escena a la que le indiquemos que vaya.

Nuestra misión será interactuar con todos los objetos que encontremos, principalmente pinchando sobre ellos y almacenándolos en un elemento común a todas las Aventuras Gráficas, el Inventario. Siempre portaremos objetos que deberán ser usados con otros objetos de la pantalla, con otros objetos del inventario, o ser entregados a un personaje de la aventura. Otro elemento diferenciador de las aventuras gráficas es el hecho de que podremos entablar conversaciones con los personajes que nos encontremos. Estas conversaciones por supuesto están muy limitadas a posibles frases entre las que elegir, de modo que consigamos recabar información para completar la aventura. Las Aventuras gráficas han evolucionado desde finales de los años 80 (*Maniac Mansion*, Lucas Arts 1987) hasta nuestros días, primero ofreciendo todos los textos de la aventura locutados por actores profesionales con la popularización de los CD-ROM a mediados de los 90, y más tarde incorporando las nuevas funcionalidades 3D que

ofrecen los equipos actuales. Las Aventuras gráficas que actualmente se encuentran en el mercado han evolucionado hasta cambiar completamente la forma de jugar al usar entornos virtuales en 3D, sin embargo las características fundamentales no han cambiado.

Figura 4.16. Maniac Mansión

Ejemplos actuales de Aventuras Gráficas en 2D son el juego español *Runaway* y en 3D fue pionero el *Monkey Island 4* y tremendamente popular la serie *Broken Sword.*

Como se puede deducir por las características de este tipo de juegos, serán quizás los menos apropiados para ser portados a teléfonos móviles. Es fundamental el uso de ratón, se apoyan en recursos gráficos y sonoros demasiado "pesados" para los teléfonos móviles y las Redes GSM y GPRS del momento. Sin embargo para PDAs ya se encuentra una novedosa aplicación que es capaz de emular en la plataforma Pocket PC el sistema SCUMM (Script Creation Utility for Maniac Mansion) sobre el que corrían la mayoría de las aventuras gráficas de los años 90. Esta maravilla se llama *Pocket ScummVM* (www.scummvm.org). Por lo tanto como se comentó anteriormente, este tipo de juegos podría ser viable en los nuevos teléfonos móviles como el Sony-Ericcsoon P800 y P900, que son realmente híbridos entre PDAs y teléfonos móviles.

4.1.10 3D:

Quizá uno de los hitos en la Historia de los Vídeojuegos fue la aparición el 5 de Mayo de 1992 de un juego que dejó a todos los usuarios de PC boquiabiertos, este juego fue el *Wolfenstein*. El precursor de los llamados ***Juegos 3D en primera persona*** o ***3D Action Games***. Aunque en los tiempos del Spectrum ya existieron juegos que utilizaban mallas vectoriales para representar mundos en 3D (***Freescape games***), éstos eran tan esquemáticos que resultaban confusos de jugar. El que más éxito alcanzó fue un magnífico juego aparecido en 1990 llamado *Castle Master*. Este juego era una aventura gráfica en 3D y fue probablemente el primero en permitirnos movernos por un mundo virtual en 3D, y precursor de otro juego muy avanzado para su época como fue el *7th Guest*. Sin embargo, ninguno de estos juegos llegó al público de forma tan masiva como lo hizo el mítico *Doom*. Tanto es así, que desde su aparición se creó un género llamado ***Juegos tipo Doom*.** La única gran novedad que aportaron respecto al *Wolfenstein* fue que ahora se usaba el ratón para apuntar el arma, y no sólo las teclas cursoras. Y es que como hemos mencionado antes el verdadero pionero fue el *Wolfenstein*. Fue el primero en mostrar gráficos 3D con texturas bastante realistas, (que eran renderizados por software ya que no existía ningún apoyo de las tarjetas 3D como ocurre actualmente).

Pero lo más importante es que fue el primero en mostrar una perspectiva de cámara en que sólo se veía el arma del protagonista, consiguiéndose una inmersión en el juego sin precedente hasta esa fecha. Desde la aparición de *Doom* y sus sucesores *(Quake, Half Life*, etc.) se desató una "fiebre" por los juegos en 3D que revolucionó el mercado del entretenimiento, provocando que los fabricantes de Hardware y Sistemas Operativos tuvieran como principal meta, crear plataformas donde los usuarios pudieran jugar a los cada vez más espectaculares juegos en 3D. En cierto modo, han colaborado al desarrollo de la tecnología actual, ya que cada vez demandaban más recursos de las máquinas y éstas se quedaban anticuadas antes, con lo que los usuarios adquirían nuevo hardware cada menos tiempo, con el consiguiente abaratamiento de costes.

Figura 4.17. Castle Master

Figura 4.18. Wolfenstein

En el mismo año en el que aparecía el *Wolfenstein*, una empresa francesa llamada Infogrames lanzaba al mercado un juego revolucionario, el *Alone in the Dark*. Era un juego tipo perspectiva isométrica en el manejo del personaje, pero esta vez con un motor 3D real. La acción transcurría en una mansión encantada, teniendo como novedad técnica que la cámara se encontraba en un sitio distinto en cada estancia de la casa. De modo que el manejo no resultaba sencillo, pero resultaba gráficamente muy atractivo, ya que en todo momento veíamos perfectamente las acciones que realizaba nuestro personaje, como si fuera una

película. Por esa razón a este tipo de juegos se les llamó ***Juegos en Tercera Persona***.

Figura 4.19. Alone in the Dark

No mucho después de que juegos realmente violentos como el *Quake* mantuvieran ocupados a buena parte de los adolescentes, empezaron a surgir juegos en 3D en los que la perspectiva del jugador permitía que viéramos a nuestro personaje desde atrás. De este modo en 1996 vio la luz otro juego que ha sido imitado hasta la saciedad, el *Tomb Raider*. Mucha gente habla de ***juegos tipo Tomb Raider*** como una variedad de juegos con características comunes.

Hemos puesto todos los juegos en 3D en un mismo saco, no porque dentro de ellos no haya más clasificaciones, sino porque la dificultad técnica de desarrollar este tipo de juegos, estriba en el motor 3D. Tanto más si lo que se pretende es portarlos a plataformas móviles, como es nuestro caso. JAVA 2ME en el momento de realización del presente trabajo, no proporciona ningún tipo de facilidad para el desarrollo de juegos en 3D, y probablemente pase mucho tiempo hasta que esta situación cambie. Por tanto, si pretendemos realizar un juego para móviles en 3D usando tecnología JAVA, tendremos que crear desde cero un motor 3D propio. Y saber que éste no podrá ser muy complejo, ya que una parte de los recursos será utilizada por la máquina virtual Java. Además JAVA 2ME no tiene los tipos de

datos primitivos *float* ni *double*, fundamentales para un óptimo motor 3D. Sin embargo, se están haciendo grandes esfuerzos por parte de los fabricantes de Hardware para que la aceleración 3D llegue a los terminales móviles. Muy recientemente, las dos principales firmas de tarjetas aceleradoras, ATI y Nvidia, han sacado al mercado sendos procesadores gráficos para teléfonos móviles y PDAs, la ***ATI ImageOn 3200*** y la ***nVIDIA GoForce 4000*** respectivamente. No obstante, estos procesadores están más orientados a la aceleración 2D de imágenes de vídeo, que a 3D.

Aunque para JAVA 2ME los juegos en 3D son todavía prohibitivos, no es así para los juegos nativos para Symbian. Para la serie 60 de Nokia, existen juegos en 3D realmente espectaculares como es el: *Interstellar Flames*, o el mítico *Doom*. Si se posee un Nokia NGage, se puede jugar a versiones del *Tomb Raider* o del *Fifa 2004* que nada tienen que envidiar a las versiones de PC.

Figura 4.20. Tomb Raider Nokia N-Gage

4.2 Limitaciones de la Plataforma:

Cuando hablamos de limitaciones de la plataforma, nos referimos no sólo a las limitaciones que tienen los teléfonos móviles en comparación con los ordenadores de sobremesa, sino también a las de JAVA 2ME en comparación con JAVA 2SE. A continuación se enumeran dichas limitaciones:

4.2.1 Energía

Es el principal problema de cualquier aparato electrónico, y el causante de forma directa o indirecta, del resto de limitaciones. Es éste un campo de continua investigación, ya que cada vez los aparatos electrónicos son más complejos y demandan más recursos energéticos para funcionar. Aunque en la actualidad con la popularidad de las baterías de Litio los terminales móviles han experimentado un importante incremento de autonomía, debemos siempre tener en consideración a la hora de programar nuestros juegos las limitaciones energéticas. Por ejemplo, no deberemos abusar de la vibración pues acabaremos con la batería del teléfono en poco tiempo. Además, que el juego use vibración o sonido, deberían ser opciones configurables. Y es que el altavoz del teléfono es otro gran consumidor de energía.

4.2.2 Pantalla

Hay casi tantas variantes como modelos de teléfonos en el mercado. La principal diferencia es el **número de colores** que pueden representar. Desde monocromo hasta los 65.535 colores de los terminales más actuales. Hay que tener especial cuidado cuando desarrollemos juegos susceptibles de ser ejecutados en pantallas monocromo o de escala de grises, ya que si los gráficos no están especialmente adaptados, lo más probable es que sean tan confusos que no se pueda ni jugar. La **resolución** de la pantalla será otro factor importantísimo a tener en cuenta. Aunque en Java 2ME podemos utilizar controles para construir formularios que se verán correctamente en cualquier pantalla, cuando hacemos un juego, al final siempre tendremos que usar la clase *Canvas* (lienzo).

Y el uso de ésta está fuertemente ligado a la resolución de la pantalla. Para que nuestro juego fuera lo más portable posible, lo ideal sería que todos los gráficos que utilice fueran escalables, de modo que se pudiera adaptar el juego a cualquier tamaño de pantalla. Esto sólo será posible en contadas ocasiones, así que tendremos que adaptar nuestro juego a cada resolución. Por lo tanto en este tipo de desarrollos será de vital importancia que nuestro código tenga una buena estructura que aísle todo lo posible, la lógica del juego de las rutinas gráficas. Otro aspecto a tener muy en cuenta es la **velocidad de refresco** de la pantalla. Si estamos creando un juego de acción en el que por ejemplo usemos *scrolls* rápidos, deberemos probarlo sobre todos los terminales que tenemos como objetivo, pues habrá teléfonos que aún capaces de ejecutar nuestro programa sin problemas, la baja calidad de su pantalla a color, mostrará los gráficos en movimiento tan borrosos que serán injugables.

4.2.3 Teclado

Un factor determinante para el éxito de un juego es su jugabilidad. Y ésta depende directamente de los controles que el dispositivo nos proporcione. Lamentablemente el único móvil actualmente en el mercado concebido para jugar,

es el Nokia N-Gage. El resto de los móviles en el mejor de los casos, no están preparados nada más que para movernos en las cuatro direcciones. Por si esto fuera poco, además los teclados no suelen tener una rápida respuesta, y lo que es aún peor, cada móvil tiene una disposición distinta de las teclas. La disposición puede llegar a ser tan problemática que haga imposible el manejo del juego en determinados modelos. Especial atención en este aspecto merece el Nokia 3650, cuyo teclado se encuentra en disposición circular. Por lo tanto no sería mala idea permitir al usuario configurar los controles a su gusto.

Por otra parte, en lo referente a las facilidades de Java 2ME para gestionar las pulsaciones de teclado, son un tanto escasas pero suficientes. Por ejemplo para detectar la pulsación continuada de una tecla, deberemos hacerlo a la antigua usanza, almacenando dicha tecla en una variable.

4.2.4 Tamaño

Las limitaciones de tamaño se refieren al tamaño que ocupa el archivo .JAR final del juego listo para distribuirse. Aunque el espacio de almacenamiento disponible en los móviles suele ser muy limitado, éste no será nuestro principal problema. La razón por la que no deberemos superar un tamaño límite que puede estar entre los 90 y 100Kb., está relacionada con el proceso de *deployment* o de distribución de los juegos para móviles. El **modelo de negocio** actual de la venta de aplicaciones para móviles, consiste en que nosotros como desarrolladores llegamos a un acuerdo con la operadora, de modo que ella se encarga de hacer accesible nuestro juego a sus clientes y de la facturación. A cambio, la operadora se queda con un porcentaje por cada descarga del juego. A esta forma de distribución de las aplicaciones Java se la llama **OTA (Over-the-Air Provisioning)**. Por lo tanto nuestro fichero deberá viajar por una red GSM o por una GPRS. Redes que tienen una tasa de transferencia muy baja. Eso implica que el cliente que compre nuestro producto deberá acarrear con los gastos de la transmisión del fichero .JAR hasta su móvil. Aunque los terminales más potentes

podrían soportar juegos que ocuparan más de 1Mb sin demasiados problemas, no habría ningún cliente que se descargara dicha cantidad de datos, por lo menos con los precios actuales. Además, las operadoras imponen un límite de tamaño máximo para **OTA**, que deberemos conocer de antemano a la hora de crear nuestro juego.

Por poner un ejemplo práctico, la serie 40 de Nokia impone un límite de 64Kb. Para el fichero JAR, y de 5Kb. Para el fichero JAD. Las aplicaciones Java 2 ME se pueden instalar también usando un puerto de infrarrojos, Bluetooth o mediante USB. Sin embargo, actualmente no existe ningún modelo de negocio que permita el aprovechamiento de estos modos de instalación, ya que el usuario necesitaría tener un ordenador. Aunque el problema fundamental es que estos métodos facilitarían la piratería de los programas.

El modelo de negocio novedoso que ha puesto en marcha Nokia con la Nokia N-Gage, consiste en distribuir los juegos en SD cards (http://www.forum.nokia.com/main/0,6566,010_40_10,00.html). Equiparando el negocio al de la vídeoconsolas. Aunque la mayoría de los juegos en este soporte son nativos de Nokia, también los hay hechos en Java. Gracias a este soporte no tendremos la limitación de tamaño comentado en este apartado.

4.2.5 Memoria de ejecución

Aunque nuestro programa no ocupe mucho espacio en disco, puede ser capaz de acabar rápidamente con la memoria de ejecución del móvil. Si por ejemplo nos dedicamos a instanciar objetos indiscriminadamente sin liberarlos explícitamente, es más que probable que veamos el fatídico mensaje de *Memoria Insuficiente* y nuestro juego "muera" matando en ocasiones hasta al propio sistema operativo del teléfono. Y es que los procesadores que suelen ejecutar Java 2ME son tan poco potentes que no se pueden permitir el "lujo" de llevar una gestión minuciosa de los objetos en memoria que ya no se van a usar, como hace Java 2SE. Eso significa

que el llamado recolector de basura o *Garbage Collector* sólo sea llamado en contadas ocasiones, y debamos nosotros explícitamente invocar al método *System.gc()* de vez en cuando. Cada fabricante de terminales decide el tamaño de esta memoria de ejecución, en función de las prestaciones del móvil. Sin embargo hay un mínimo de 64Kb. de memoria de ejecución que el estándar CLDC obliga que tengan todos los terminales. Por ejemplo la serie 60 de Nokia tiene cerca de 4Mb de memoria de ejecución.

4.2.6 Sonido

MIDP 1.1 no contempla nada más que el uso de *ring tones.* Por lo tanto a no ser que usemos una API específica del fabricante que permita reproducir sonidos polifónicos, nuestro juego estará limitado al uso de unos *beeps* en los que podremos programar la frecuencia y duración de éstos. Además la mayoría de los modelos tienen unos altavoces de baja calidad. Cosa que tendremos que tener en cuenta, pues será inútil usar sonidos digitales de alta calidad. En este aspecto las cosas están cambiando y teléfonos como los Sony-Ericsson y los Samsung tienen un sonido estéreo de gran calidad. No así los Nokia, en los que no merecerá mucho la pena que nos esmeremos demasiado en el aspecto sonoro.

4.2.7 Procesador

Los procesadores de los teléfonos móviles son por lo general muy lentos y nuestros programas no deberían demandar demasiados recursos de ellos. Más aún si estamos desarrollando en Java, porque aparte de los recursos que demande el sistema operativo, que no son pocos, se estará ejecutando la máquina virtual Java, y encima de todo eso, se deberá ejecutar nuestro juego. Para complicar un poco más las cosas, los fabricantes de móviles no mencionan el tipo de procesador que utilizan sus modelos, ni mucho menos la frecuencia de éstos. Además, la diferencia de rendimiento entre distintos modelos son muy variables, como ejemplo, en la serie 60 de Nokia, hay una diferencia notable de rendimiento entre el Nokia 7650 y el Nokia 3650, siendo este último más rápido que el anterior.

Por lo tanto, como siempre, deberemos probar nuestros juegos en los modelos objetivo, pues un juego que es capaz de "mover" un Nokia 3650 por ejemplo, en un Nokia 7650 puede ser injugable.

4.2.8 Ausencia de float y double

Los procesadores de los teléfonos móviles son muy sencillos, tan sencillos que normalmente no tienen ni unidad de coma flotante. Es por esta razón que Java 2ME no dispone de los tipos de datos primitivos *float* y *double*. Y sin ellos es imposible en principio hacer cálculos trigonométricos, razón por la cuál han desaparecido métodos de la clase ***Math*** como ***cos()*** y ***sin()***.

4.2.9 Hilos sin método stop()

Otro de los múltiples recortes de Java 2ME es la supresión de métodos en la clase *Thread.*

4.2.10 Red de comunicaciones de banda estrecha

Como ya se ha mencionado anteriormente al hablar del tamaño final del juego, la red GSM y GPRS es lenta, cara e inestable. Debemos tener en cuenta esto si pretendemos que nuestra aplicación envíe o reciba datos. El usuario de nuestro programa deberá estar siempre bien informado del uso que se hace de la red, pues el uso de este recurso cuesta dinero. Las operadoras españolas acaban de empezar a comercializar UMTS. Banda ancha de momento sólo al alcance de los bolsillos de empresas. Por lo tanto, con la red actual, para nuestro juego puede ser factible por ejemplo bajarse la lista de las 10 mejores puntuaciones alojada en un servidor, o quizá enviar o recibir un par de imágenes de baja resolución.

4.3 Presupuesto, Equipo Humano y Timing

Originariamente, el desarrollo de un juego era el resultado del trabajo de un único programador o como mucho de un grupo de no más de 10 programadores. Sin embargo, los presupuestos que se manejan actualmente para desarrollar software de entretenimiento, nada tiene que ver con el resto del software. Las causas pueden estar en un aumento inmenso de la competencia en este sector, que cada vez ponía el listón más alto para destacar entre los miles de juegos que salían al mercado cada año, y en un aumento paulatino de las posibilidades gráficas y sonoras del hardware y, en consecuencia, de la demanda de los jugadores.

Es muy significativo el hecho de que a principios de los años 90, muchas películas de Hollywood orientadas a los jóvenes se “llevaban” en forma de vídeojuego a la pantalla de los PCs (*Batman The Game)*, mientras que en los últimos años hemos asistido a lo contrario, Juegos de PC que posteriormente se convierten en películas multimillonarias de Hollywood (*Street Fighter, Mortal Kombat, Tomb Raider*, *Resident Evil...*).

En consecuencia, los presupuestos para el desarrollo de juegos de ordenador y consolas se asemejan ahora al de una película de Hollywood. Sólo al alcance de grandes compañías y multinacionales. Centrándonos ahora en el nuevo y emergente mercado de Juegos Java, es de destacar que los presupuestos que se manejan, nada tienen que ver con el mercado del entretenimiento digital actual. Más bien se puede hablar de una "vuelta al pasado", pues un Juego Java comercial, puede desarrollarse en un tiempo razonable por un único programador. Este hecho es muy beneficioso para estudiantes de informática, y pequeñas empresas de software, pues podrán competir en igualdad de condiciones con las grandes empresas que copan el mercado de vídeojuegos, "intocable" sin grandes inversiones. Y es que es difícil que un Juego Java supere presupuestos de 100.000€, cantidad ridícula para el mercado de vídeojuegos de consola y PC.

El equipo humano ideal debería estar formado por dos o tres personas. Un programador

Java a ser posible con conocimientos de las técnicas clásicas de programación de vídeojuegos. Un diseñador gráfico, que a parte de manejar programas de retoque gráfico, tuviera conocimientos de animación 2D y/o 3D. Una tercera persona podría estar especializada en el apartado sonoro, aunque se pueden adquirir sonidos de inmensas librerías en Internet que nos permitirían ahorrarnos este técnico de sonido. El tiempo de desarrollo de un Juego Java puede estar entre 3 y 6 meses. Dependiendo claro está, de la complejidad de éste.

4.4 Estudio de la tecnología en el mercado de dispositivos móviles

En el estudio de la tecnología de juegos en dispositivos móviles, debemos conocer donde enmarcaremos dicha tecnología. En este sentido, nos enfocaremos en los terminales Nokia, y más concretamente a los de la Serie 60, por varios motivos:

- ✓ Por cuota de mercado, ya que Nokia actualmente es el líder indiscutible del sector.
- ✓ Porque son los que permiten actualmente los Juegos Java más espectaculares.
- ✓ Porque los poseedores de este tipo de terminales son los mayores demandantes de Juegos Java.
- ✓ Porque la llamada Serie 60 es un estándar abierto que pueden implementar otros fabricantes que no sean Nokia.
- ✓ Porque los desarrolladores serios de Juegos Java adaptan sus juegos a sus terminales objetivo, para sacar el mayor partido de ellos. De modo que siempre habrá que acabar utilizando una API propietaria de un fabricante, que en este caso será la de Nokia, pero siguiendo procedimientos similares se puede usar la API de Siemens por ejemplo.

Lo primero que debemos saber como programadores, es sobre qué Sistema Operativo vamos a programar. El "Windows" en telefonía móvil se llama **Symbian OS** (http://www.symbian.com). Y decimos el "Windows", no sólo porque sea un sistema operativo, sino porque está tan extendido en los teléfonos móviles, como Windows en los ordenadores personales. Actualmente la mayoría de los fabricantes de teléfonos móviles (Nokia, Sony-Ericsson, Siemens, Samsung y Panasonic, entre otros), instalan Symbian OS en sus terminales. Aunque cambian la interfaz gráfica, y parecen Sistemas Operativos distintos, por debajo todos ejecutan el mismo Sistema Operativo. En el momento de la realización de este trabajo, Symbian OS va por su versión 9.3, que ya tienen teléfonos como el Nokia 6600.

Microsoft ha detectado un importante nicho de mercado, y ha sacado al mercado los llamados *SmartPhones*, que vienen con sistema operativo **Windows Mobile 2003 Pocket PC Phone Edition.** Actualmente en España no están muy extendidos, pero telefónica en su gama más alta de terminales para empresa, ofrece el TSM500 que lleva este Sistema Operativo. Es una incógnita si Microsoft conseguirá llevarse el mercado como ha hecho en el mundo PC, sobre todo porque el resto de fabricantes ha hecho frente común contra el gigante azul, e incluso empresas de software de desarrollo como Borland, han cerrado filas en torno a Symbian OS. Sin embargo, es de esperar que con las facilidades de programación usando .NET, soportado por los *SmartPhones*, y el inmenso poder de Microsoft, al final se lleven el gato al agua.

Existen actualmente dos tecnologías para desarrollar aplicaciones para teléfonos móviles con sistema operativo Symbian, **Java2ME** y **C++** [PRIETO04]. Debemos tener claro que desde el momento en el que decidimos emplear tecnología Java para nuestras aplicaciones nos habremos atado no sólo a las ventajas sino también a las desventajas de usar ***código interpretado*** . Éstas son principalmente ***baja velocidad de ejecución*** y ***limitaciones funcionales.*** Para

entender estas desventajas hay que saber distinguir entre ***código interpretado*** y ***código nativo.***

El ***código nativo*** es el código que obtendríamos al compilar un programa en **C++**. Es el código máquina que puede ser ejecutado por el sistema operativo del teléfono móvil.

El ***código interpretado*** es el código que obtendríamos al compilar un programa en **Java**. Ese código debe ser traducido a código máquina en el momento de su ejecución, por la ***Máquina Virtual Java***. Esa *traducción* es la responsable de la baja velocidad de ejecución mencionada anteriormente.

Cuando hablamos de ***limitaciones funcionales*** nos referimos a que sólo dispondremos de los recursos que la implementación de la máquina virtual Java nos permita utilizar. La máquina virtual Java por tanto nos *aísla,* en la llamada ***sandbox*** (caja de arena), de las peculiaridades del teléfono que estamos programando. Razón por la cuál un programa Java no podrá por ejemplo, usar la conexión por infrarrojos del teléfono, a no ser que el fabricante nos proporcione una API a tal efecto. Por ejemplo, la especificación **MIDP 1.0** de **Java2ME,** sólo permite el uso de tonos telefónicos (*ring tones*) en cuanto a sonido se refiere. Sin embargo los fabricantes de terminales con capacidad para reproducir sonidos polifónicos o MIDI, como Nokia o Sony-Ericsson, permiten descargar desde la web las APIs que entre otras, nos proporcionan funciones para que nuestros programas puedan sacar el máximo partido de sus terminales.

El cuadro siguiente muestra una comparativa entre las dos tecnologías comentadas:

	J2ME	SYMBIAN OS
Tamaño de aplicación permitido	Decenas de Kb.	Varios MB.
Estándar Abierto	Sí	Sí
Deployment (Despliegue)	Grande	Más pequeño
Soporte de Varios Fabricantes	Sí	Sí
Instalación OTA (Over-the-Air)	Sí*	Sí[1]
Ejecución de forma Nativa	No	Sí
Lenguaje de programación	JAVA	C++
Comunicación con Servidores Remotos	Sí	Sí
Animación en 2D	Sí	Sí
Animación en 3D	No	Sí
Mostrar Vídeos	No[2]	Sí, si lo permite el terminal
Audio MIDI	Sí*	Sí
Audio de Alta Calidad	Normalmente No	Sí
Acceso a SMS	Normalmente No[3]	Sí (Además MMS si lo soporta el terminal)
Acceso a puerto de Infrarrojos y Bluetooth	No	Siempre que lo tenga el terminal
Acceso a la agenda, Calendario, etc	No	Sí
Marcar un teléfono	No	Sí
Complicaciones de Desarrollo Multiplataforma	Sí	Sí

* Excepto para teléfonos antiguos.
[1] Sin embargo, Las aplicaciones de Symbian OS son normalmente tan grandes que es imposible la distribución OTA.
[2] J2ME puede reproducir vídeos en teléfonos que soporten la JAVA Mobile API (como el Nokia 3650 y posteriores).
[3] Acceso a los SMS desde J2ME es posible usando la Nokia SMS API (Soportada en los 3410) o la Wireless Messaging API (WMA, soportada por los Nokia 3650 y posteriores).

4.4.1. La serie 60 de Nokia

La Serie 60 de Nokia (http://www.series60.com), es una línea de productos guiada por las prestaciones en contraposición a otras de Nokia guiadas por precio y tamaño del terminal, como por ejemplo la Serie 40. Conocer las características técnicas de los terminales objetivo, es de vital importancia a la hora de desarrollar aplicaciones para móviles ya que, aunque la tecnología de programación es la misma, las plataformas son muy distintas entre sí. Incluso entre los distintos modelos de una misma serie se encuentran diferencias importantes en cuanto a

recursos y disposición del teclado. Por ejemplo, lo único que tiene en común la serie 60 de Nokia es el tamaño de la pantalla (**176 x 208** píxeles), una profundidad de color mínima de 4.096 colores y capacidad de conexión Bluetooth. Otros aspectos como puede ser la capacidad de proceso, o de sonido, son similares pero en absoluto idénticos. Cada modelo que aparece al mercado dentro de la Serie 60 es superior al anterior, y no nos deberemos guiar tanto del número de modelo como del orden de aparición en el mercado. Por ejemplo, el Nokia 3650, es superior al 7650.

La Serie 60 de Nokia está compuesta en el momento de elaboración del presente trabajo por los siguientes modelos ordenados cronológicamente según orden de aparición:

7650: Primer modelo de la Serie 60. Modelo de alta gama y alto coste, Orientado a ejecutivos. Es el único modelo de la serie 60 que dispone de foto sensor (No programable en MIDP). Este modelo no se comercializó en U.S.A. De serie no lleva software para grabación de vídeo pero Nokia lo proporciona gratuitamente para descargar en su web. No tiene lector de tarjetas MMC. Tiene capacidad de sonido polifónico pero dispone de un altavoz de poca calidad. Lleva cámara incorporada.

Tamaño máximo de aplicación JAVA: 4Mb.

Memoria de ejecución máxima (Heap Size): 1,4Mb.

3650: Terminal multimedia orientado a gente joven. Viene de serie con software para grabación de vídeo y para reproducción (Real ONE player). La disposición del teclado en círculo es de vital importancia tenerla en cuenta a la hora de definir las teclas de control de nuestro juego. Tiene capacidad de sonido polifónico pero dispone de un altavoz de poca calidad. Incorpora lector de tarjetas MMC. Lleva cámara incorporada.

Tamaño máximo de aplicación JAVA: 4Mb.

Memoria de ejecución máxima (Heap Size): 1,4Mb.

N-Gage: Es el primer móvil orientado al mercado del entretenimiento, puede reproducir MP3 y es el primer móvil en incorporar capacidades de aceleración 3D. Es competencia directa de consolas de videojuegos portátiles como la Gameboy. Numerosas compañías desarrolladoras de software de entretenimiento han portado ya los juegos más populares a esta plataforma. Títulos como Tomb Rider y Sonic the Hedgehog ya se encuentran en el mercado, con una calidad en cuanto a gráficos y sonido, muy similar a la de las antiguas consolas de 16 bits como la Megadrive de Sega. Los juegos se distribuyen en tarjetas MMC. Este modelo no incorpora cámara.

Tamaño máximo de aplicación JAVA: 4Mb.

Memoria de ejecución máxima (Heap Size): 2,8Mb.

6600: Es el primer teléfono de la comentada anteriormente *Developer Platform 2.0 for Series 60.* Lleva la versión 7 del sistema operativo Symbian que a simple vista no se diferencia mucho de la versión anterior. Su pantalla es de 65.535 colores. La mejora más necesaria que se echaba de menos en modelos anteriores ha sido en lo referente al sonido. En este terminal se escuchan los sonidos polifónicos con gran calidad, además incorpora un nuevo tipo de sonidos llamados **TRUE TONES** basados en la tecnología *Adaptive Multi-Rate Wideband (AMR-WB)*. Esto es importante ya que es la tecnología de audio estándar para los teléfonos de tercera generación, aprobada por el organismo 3GPP. Es también destacable la posibilidad de zoom de la cámara incorporada, sin haber variado la resolución ni la calidad de la misma respecto a modelos anteriores. Lo más importante de este teléfono es que es el primer Nokia en soportar **MIDP 2.0**. que facilita la programación de juegos. La versión para el mercado americano de este modelo se llama 6620.

Tamaño máximo de aplicación JAVA: 4Mb.

Memoria de ejecución máxima (Heap Size): 3Mb.

3660: Es prácticamente idéntico al modelo 3650 salvo que la disposición del teclado ha cambiado de la disposición circular a la típica de cualquier teléfono móvil. Además su pantalla es capaz de mostrar 65.535 colores. La versión americana de este modelo se llama 3620.

Tamaño máximo de aplicación JAVA: 4Mb.

Memoria de ejecución máxima (Heap Size): 1,4Mb.

7610: Es el primer teléfono de Nokia en incorporar una cámara de 1 Megapíxel. Además es novedoso también su puerto USB Pop-PortTM que permite una rápida conectividad con un PC. Por todo lo demás, es similar al modelo anteriormente comentado. **Tamaño máximo de aplicación JAVA:** 4Mb.

Memoria de ejecución máxima (Heap Size): 8Mb.

N-Gage QD: Es el nuevo modelo de la consola de juego N-Gage, ofrece una Buena connexion inalámbrica para juegos en red. Posee un alto rendimiento para el juego en 3D, A diferencia de las demás consolas de juegos, N-Gage QD provee la posibilidad de juegos multijugador sobre Bluetooth y GPRS. Conexión para e-mail mediante IMAP4, POP3, SMTP, MIME2.

Tamaño máximo de aplicación JAVA: 4Mb.

Memoria de ejecución máxima (Heap Size): 8Mb.

Nokia 6682: El Nokia 6682 imaging smartphone está diseñado para mejorar el entretenimiento multimedia. Cuenta con una pantalla de 2" y 262,144 colores. Sonido stereo, reproductor MP3/AACr, grabación de videos, camara fotográfica de 1.3 megapixeles. Soporta MIDP 2.0.

Otras espcificaciones: Visor de archivos Word/Powerpoint/Excel y Adobe pdf, PoC, RealOne Player. Tecnología Java: CLDC 1.1, MIDP 2.0, Nokia UI API, Wireless Messaging API (JSR-120), Mobile Media API (JSR-135), Bluetooth API (JSR-82 No OBEX), FileConnection and PIM API (JSR-75), Mobile 3D Graphics API (JSR-184), Java Technology Wireless Industry rel. 1 (JSR-185).

Tamaño máximo de aplicación JAVA: Alojamiento dinámico en memoria.

Memoria de ejecución máxima (Heap Size): Alojamiento dinámico en memoria.

Nokia N93: El Nokia N93 es la primera cámara en el mercado con 3.2 megapixeles con autofoco , zoom óptico 3x y cámara de video digital.

Graba y reproduce de forma fluida películas tipo DVD de MPEG-4 VGA a 30 fps con sonido audio estéreo.

Pantalla de 2.4" con amplio ángulo de visión de 160°: 320 x 240 píxeles, hasta 262.144 colores.

Interfaz de usuario: S60 3ª Edición con tecla de desplazamiento de 5 direcciones, teclas de selección, tecla de menú, teclas de edición y borrado, teclas de llamada y finalización. Conexiones: • LAN inalámbrica integrada (802,11 b/g) y UPnP (Universal Plug and Play) • Tecnología Bluetooth integrada v.2.0 • USB 2.0 a través de interfaz Pop-Port™ • Admite salida de TV (PAL/NTSC) • Conexión con Nokia PC Suite a través de USB, infrarrojos • Sincronización inalámbrica a distancia • Envía y recibe imágenes, secuencias de vídeo, gráficos y tarjetas de visita a través de la tecnología Bluetooth

5. Propuesta de Arquitectura de desarrollo de juegos persistentes multigujador en la plataforma de agentes Jade-Leap para terminales móviles

5.1 Contexto de la Arquitectura propuesta

El contexto en el que nos encontramos presenta fuertes restricciones en cuanto a la capacidad de almacenamiento, procesamiento, visualización e interconexión, que son muy limitados. Es decir, nos encontramos ante un dispositivo de pequeño tamaño al que le vamos a exigir unas altas prestaciones.

Debido a la utilización de los agentes móviles se pueden reducir los requisitos que debe poseer el terminal móvil porque podemos trasladar la ejecución de los programas a un ordenador de sobremesa. Por ejemplo, como la tecnología de agentes móviles realiza una menor utilización de la red el ancho de banda necesario es menor. Además la conexión puede ser más inestable debido a que los agentes utilizan conexiones asíncronas y por lo tanto la utilización de la red es más puntual.

5.2 Arquitectura

La arquitectura de aplicaciones de agentes que vamos a proponer implica que en nuestro terminal móvil tengamos una plataforma capaz de ejecutar y lanzar agentes hacia otros elementos de red, que pueden ser otros terminales móviles o elementos de la red fija. Además debido a las limitaciones de almacenamiento será necesario que algunos agentes residan un tiempo limitado en el terminal, por lo tanto será clave el control y gestión del número y tipo de agentes que residen en los móviles.

Existen estudios realizados sobre la posibilidad de utilizar los estándares existentes de plataformas de agentes móviles en el contexto de sistemas de comunicaciones móviles de tercera generación. En concreto, en el capítulo 4 se analiza brevemente la utilización del estándar MASIF (Mobile Agent System Interoperability Facility) del OMG (Object Management Group) para construir una plataforma de agentes orientados a la provisión de servicios personalizados.

Nuestra propuesta consta de una arquitectura basada en agentes móviles de la que forman parte los agentes *Gestor*, *ControladorPartida*, *Calculador*, *VisualizadorJugador* y *Jugador*. Los cuatro primeros funcionan en el Dispositivo Servidor y el último en el Dispositivo del Usuario que se inicia cuando se arranca el sistema. Ver figura 5.1.

- **Agente Gestor**, agente estático que se encarga de gestionar las comunicaciones con el agente Jugador y el agente Controlador de la partida. Tiene como tareas principales:
  - Registrar las partidas que organiza en el agente DF para que los jugadores puedan encontrarlos.
  - Buscar a los agentes Controladores de las Partidas a través del DF.
  - Registrar a los jugadores en las partidas.
  - Se ocupa de la organización de cada partida.

- **Agente Controlador de la Partida**, agente estático que cumple las siguientes tareas:
  - Registra a los jugadores para cada partida.
  - Es informado de la localización de cada jugador y de la última acción (evento) que realiza.

- Recalcula el nuevo estado de cada jugador en función de su localización, estado actual y acción que realiza.

- Informa al agente de cálculo los detalles necesarios de cada jugador para el cálculo de su puntaje y número de vidas restantes.

- Informa al agente visualizador de los datos de la escena de juego de cada jugador.

- Debe registrarse en el DF para ser encontrado por el Agente Gestor.

- Acepta órdenes del Agente Gestor.

- Proporciona el resultado de la partida al Agente Gestor

- **Agente de cálculo de puntaje**, agente estático que cumple las siguientes tareas:
  - Recalcula el número de vidas de cada jugador, su puntaje a nivel de jugador y a nivel de equipo y también decide quien gana la partida.

- **Agente Jugador**, agente estático que se despliega en el dispositivo del usuario y cumple las siguientes tareas:
  - Provee la interfaz gráfica que permite al usuario registrarse y enviar la petición de acceso en la modalidad deseada (individual o cooperativa) al agente Gestor.

  - Busca partidas en DF y se inscribe al que quiere.

- Recibe información del agente Gestor sobre la partida y el controlador asignado.

- Envía al agente controlador información de la nave: localización, estado y evento activado por el usuario.

- **Agente Visualizador**, éste es un agente móvil que se traslada constantemente desde dispositivo Servidor al dispositivo del Usuario (mientras siga con vida el Jugador) para dibujar en el dispositivo servidor y mostrar en el dispositivo del usuario la escena de juego del usuario en cada instante de tiempo (parametrizable). Esta comunicación se realiza de forma asíncrona.
  - Este agente recibe información acerca de la creación de un nuevo jugador del agente controlador de la partida.

  - Dibuja (procesa) la escena de juego de cada jugador en el dispositivo servidor en base a los datos proporcionados por el agente controlador en un instante de tiempo.

  - Se desplaza hacia el dispositivo móvil para mostrar la escena de juego.

  - Existe una agente Visualizador por cada jugador registrado por el agente Controlador.

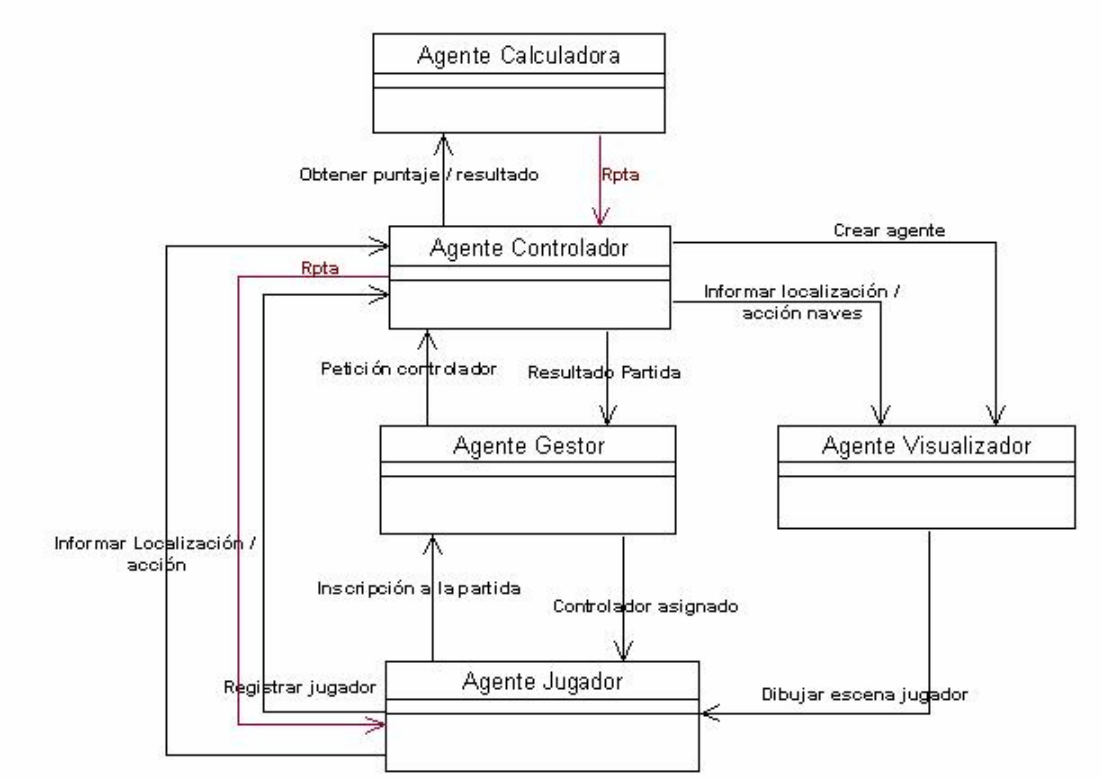

Figura 5.1. Arquitectura propuesta de una aplicación lúdica multi-agente

5.3 Caso de Estudio: Star War

Como caso práctico de estudio, analizamos en esta sección el diseño e implementación un videojuego multijugador bajo la arquitectura propuesta.

El objetivo del proyecto es conseguir la interacción entre los jugadores en tiempo real, para lo cual se ha elegido el juego StarWar, en donde se pretende eliminar a las naves enemigas de manera individualista o cooperativa. El juego inicia cuando los jugadores se suscriben en el dispositivo Servidor mediante conexión por Bluetooth a través de sus terminales móviles. En el modo de juego a nivel de cooperativo, las estrategias son un poco diferentes a las de la modalidad individual, puesto que un miembro del equipo puede tener como objetivo la defensa de la nave nodriza, mientras que otro el de derribar a la nave nodriza

enemiga o el de cubrir a otros compañeros. En la modalidad individual, la misión se basa únicamente en derribar a la nave enemiga.

El jugador que quiera entrar al juego, deberá elegir la modalidad de juego (individual o cooperativo) e introducir su seudónimo. Esta petición será procesada por el agente Gestor quien lo registrará y le dará acceso a la partida que encuentre disponible, o en caso contrario lo registrará en la cola de espera.

La acción transcurre en tiempo real, donde el jugador tiene pleno control de los movimientos y acciones de la nave que interpreta, compitiendo en un recinto virtual "enfrentándose a muerte" unos contra otros.

1.3.1 Proceso de Desarrollo

Para desarrollar esta aplicación se ha seguido la metodología de desarrollo de Sistemas Multiagente INGENIAS [GRASIA05].

El proceso de desarrollo que se ha seguido es el siguiente:

Fase Requisitos: En esta fase se realiza un Documento de Especificación de Requisitos que aporta la información necesaria para tener una visión global de la aplicación. A partir de este documento se pueden identificar las funcionalidades del mismo y se identifican los Agentes y los Casos de Uso, creando un Diagrama de Casos de Uso. Estos casos de uso se priorizan para identificar en que iteración serán desarrollados y posteriormente se detallan. Para la mejor comprensión del sistema antes descrito se incluye el Diagrama de Casos de Uso de la Figura 5.2.

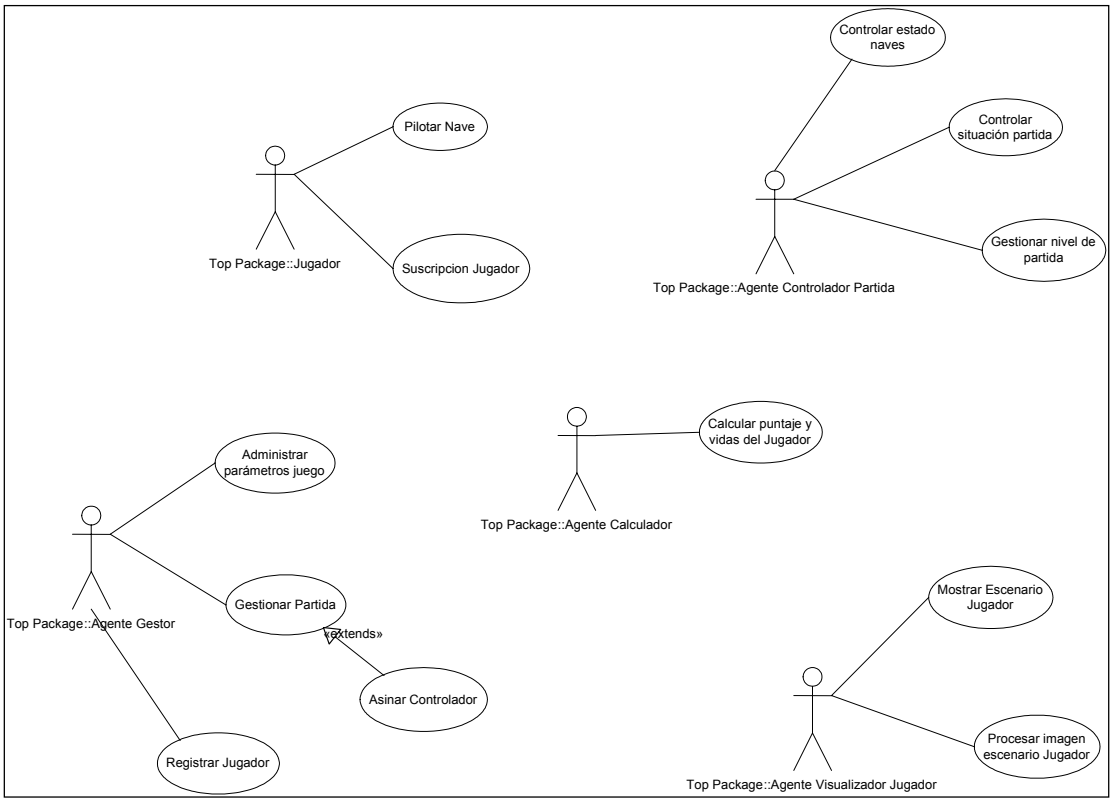

Figure 5.2. Diagrama de Casos de Uso de la aplicación

Fase Análisis: con los Casos de Uso ya identificados se procede a identificar aquellos en los cuales se produce alguna interacción entre los agentes y se asocian estos con Modelos de interacción. Tras el desarrollo de estos Modelos de comunicación se crea un primer Modelo de Organización en el cual se esboza la arquitectura del sistema e identifican los flujos de trabajo. Se genera el Modelo de Entorno y se crean los Modelos de Agente. Finalmente se crea el Modelo de Tareas y Objetivos para generar restricciones de control (los objetivos principales y la relación de las tareas con los objetivos). En la Figura 5.3 y Figura 5.4 se incluyen el Modelo de comunicación entre agentes, para una mejor comprensión del sistema.

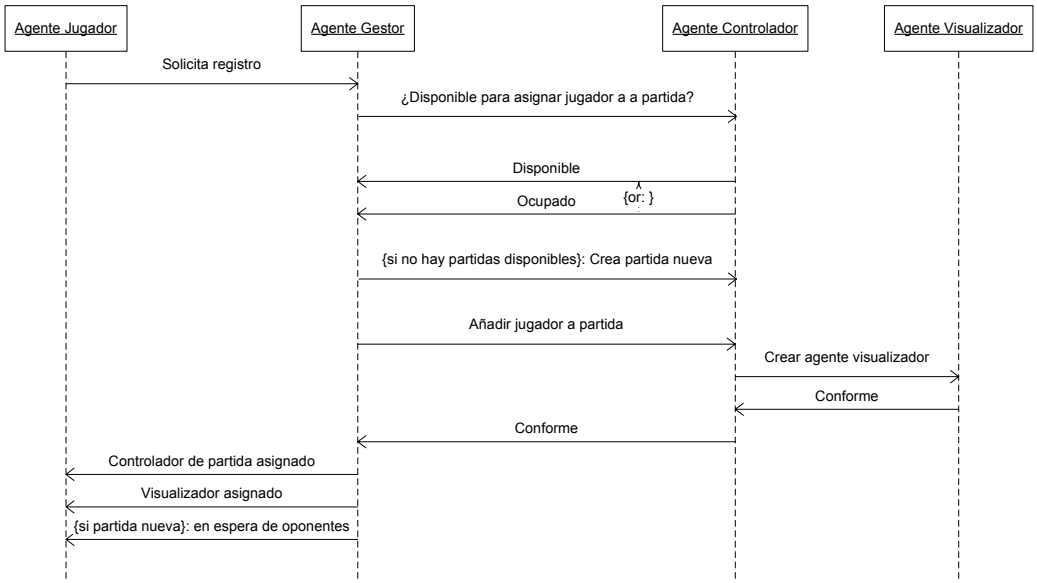

Figure 5.3. Secuencia de suscripción de un jugador en la partida

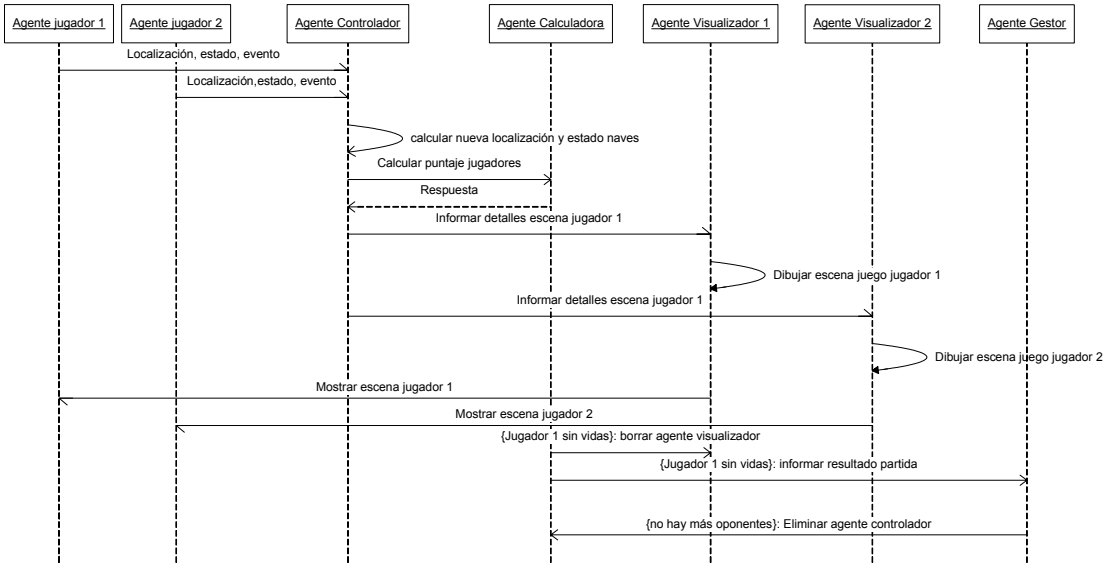

Figure 5.4. Secuencia del desarrollo de una partida

Fase Diseño: En esta fase se completa el Modelo de Organización mediante el desarrollo de los flujos de trabajo identificados durante el análisis. A continuación se completan los Modelos de Tareas y Objetivos y los Modelos de Interacción con especificaciones GRASIA de las interacciones que estos representan. Se define la ontología del sistema y finalmente se realiza el diseño de las clases que forman parte del mismo.

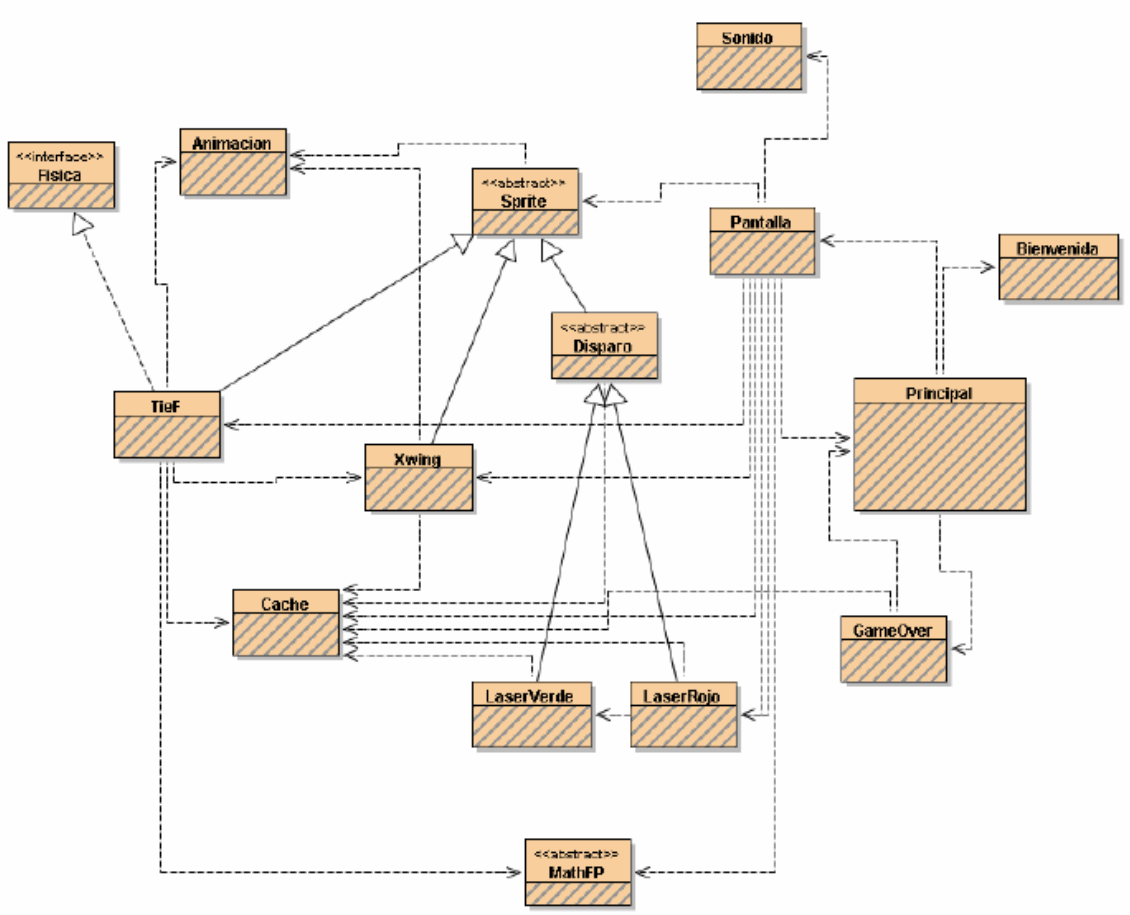

Figure 5.5. Esquema UML de clases de la aplicación

Fase de Implementación: se desarrolla el sistema a partir de la especificación del diseño y se realiza un diagrama de componentes indicando la relación entre los diferentes ficheros de código y componentes del desarrollo.

Fase Pruebas: por ultimo se realizan las pruebas del sistema con el fin de garantizar la corrección del desarrollo.

1.3.2 Implementación

En este apartado se comentan algunas decisiones tomadas durante la implementación de la aplicación y que pueden ayudar a su mejor comprensión. Principalmente se abordan cuestiones sobre la plataforma JADE-LEAP.

Como se aprecia en la figura 5.6, existe un contenedor principal en la aplicación, el cual se ejecuta en el dispositivo Servidor, que contiene al agente Gestor, a los agentes Controladores, a los agentes Calculadoras, a los agentes Visualizadores y a los agentes básicos de Gestión definidos por FIPA (DF, AMS y RMA, quienes gestionan el GUI de la plataforma JADE). Y existe un contenedor por cada dispositivo móvil que se suscribe a la plataforma de juego, en donde se ejecuta un agente personal (jugador) en cada uno de ellos. En este caso, se aprecia que existen 3 usuarios suscritos.

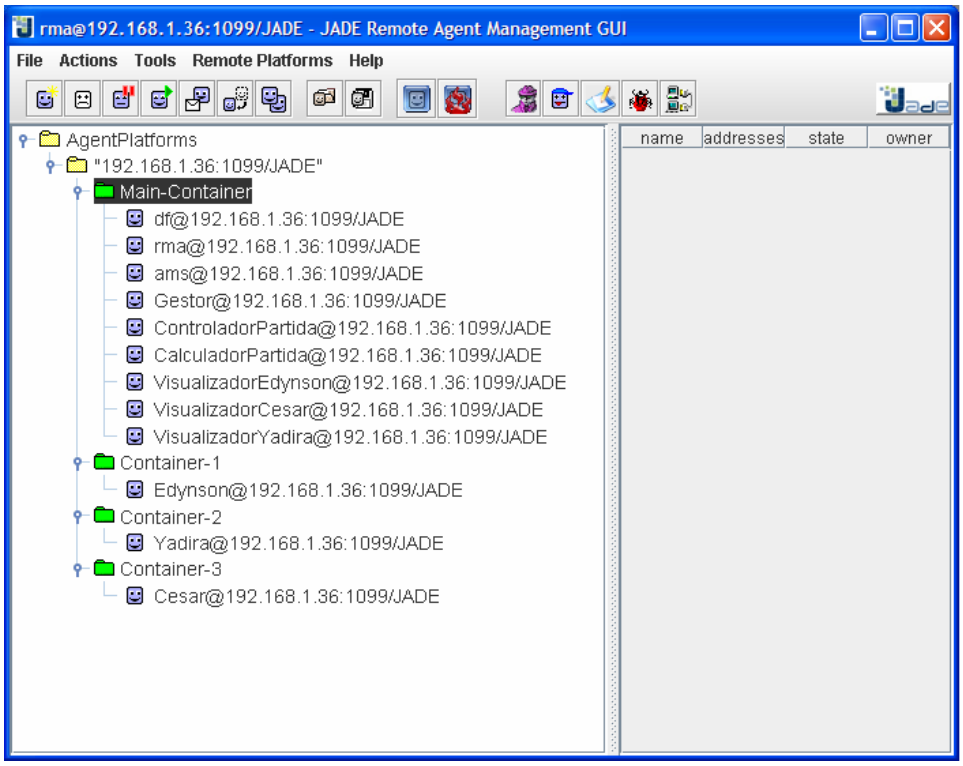

Figure 5.6. Distribución de los agentes en sus contenedores

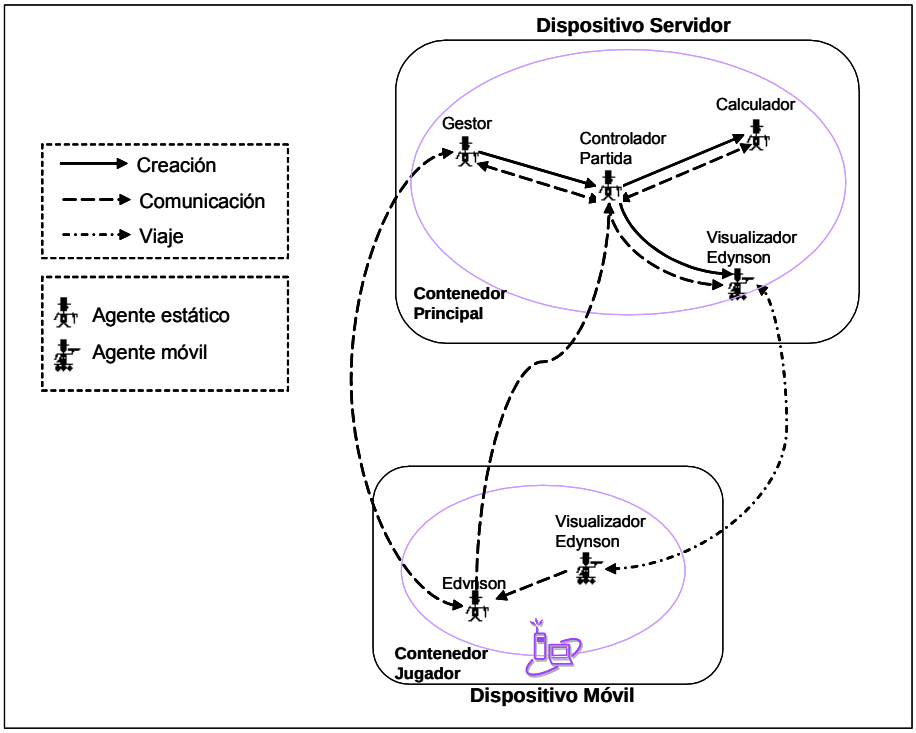

Figure 5.7. Arquitectura de despliegue de agentes en Star War

Agente Gestor:

Dentro de la plataforma JADE-LEAP los agentes son subclases de la clase Agent. Este agente al iniciarse añade un comportamiento cíclico (CyclicBehaviour) a la espera de la petición de registro de los jugadores. Al recibir la petición por parte del Agente Jugador, éste localiza a los agentes Controladores que se encuentran disponibles en ese momento mediante el agente facilitador de directorio (DF) definido en la arquitectura abstracta de FIPA. Para ello podemos usar el método DFService.search(). Una vez localizados los agentes controladores, envía un mensaje ACLMessage.CFP a cada uno de ellos en busca del primer agente Controlador disponible para la suscripción del jugador a la partida. El siguiente fragmento de código nos muestra la forma de localizar a los agentes Controladores:

```
Public class GestorAgent extends Agent {
        …
        //Lista de los agents controladores registrados
        agents private AID[] controladorAgents;
        …

        Protected void setup() {
                …
                //Adiciona un comportamiento TickerBehaviour que se activa cada segundo
                addBehaviour(new TickerBehaviour(this, 10000) {
                        protected void onTick() {
                                //actualiza la lista de agentes Controladores
                                DFAgentDescription template = new DFAgentDescription();
                                ServiceDescription sd = new ServiceDescription();
                                sd.setType("controlador");
                                template.addServices(sd);
                                try {
                                        DFAgentDescription[] resultado = DFService.search(myAgent,
template);
                                        controladorAgents = new AID[resultado.length];
                                        for (int i=0; i<resultado.length; i++) {
                                                controladorAgents[i] = result.getName();
                                        }
                                } catch (FIPAException fe) {
                                        fe.printStackTrace();
                                }

                                //Solicita suscripción de Jugador a cada controlador
                                ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
                                for (int i=0, i<controladorAgents.length; i++) {
                                        cfp.addReceiver(controladorAgents[i]);
                                }

                                cfp.setContent(esDisponible);
                                cfp.setConversationId("registro-Jugador");
                                cfp.setReplyWith("cfp" + System.currentTimeMillis());
```

```
                                        myAgent.send(cfp);
                                        ...
                                }
                        }
                }

}
```

Agente Controlador:

Este agente añade un comportamiento que se encargan de ejecutar un conjunto de acciones de manera secuencial (SequentialBehaviour) formados por subcomportamientos que se ejecutan de forma secuencial siguiendo el siguiente esquema: Primero ejecuta un comportamiento que se encarga de registrar al jugador en la partida. Después recibe en cada instante de tiempo del agente jugador información de su localización y evento que realiza. Una vez que tiene esta información de cada jugador de la partida, recalcula la situación de cada nave y envía esta información al agente Calculador quien procesa el puntaje de cada jugador. Toda esta información es enviada al agente visualizador de cada jugador. Se presenta un fragmento de este código en el anexo A.

Agente Visualizador:

El agente Visualizador hereda de la clase GuiAgent proporcionada por la plataforma
JADE-LEAP lo cual le permite tener su propia interfaz gráfica.

Se implementa la interfaz gráfica de la API propietaria de Nokia (Nokia User Interface) para aprovechar las ventajas que ofrece en cuanto a funcionalidades gráficas fundamentales para el desarrollo de juegos de buen nivel.

Este agente también añade un coportamiento cíclico (CyclicBehaviour) esperando la llegada de mensajes ACLMessage.INFORM del agente Controlador con la información necesaria para ejecutar el comportamiento de pintado de la escena del jugador. Seguidamente se ejecuta un comportamiento que se encarga de trasladar este agente del dispositivo Servidor al dispositivo del usuario y finalmente muestra la escena de la partida en la interfaz gráfica del dispositivo móvil.

```
public void run()
{

        while (running)
        {

                comienzoCiclo = System.currentTimeMillis();

                moverJugador();
                moverEnemigos();
                moverDisparos();

                repaint();
                serviceRepaints();

                time = comienzoCiclo - System.currentTimeMillis();

                        if (time < MS_POR_FRAME)
                        {

                                //De vez en cuando que suenen los TieF pasando
                                if (Cache.NumAleat(0,100) == 0)
                                {

                                        if (!Pantalla.sonidoOcupado)
                                        {
                                                sonidosStop();
                                                sonidos[SON_PASSBY].reproducir(1);
                                                sonidoOcupado=true;
                                        }

                                }

                                //Si sobra tiempo lo dedicamos a IA
                                comienzoCiclo = System.currentTimeMillis();

                                if (ultimoIA <= NUM_ENEMIGOS) ultimoIA=0;
                                else ultimoIA++;

                                enemigos[ultimoIA].IA(jugador);

                                time = comienzoCiclo - System.currentTimeMillis();

                                if ( (MS_POR_FRAME - time) > 0 )
                                {
                                        try
                                        {

                                        Thread.sleep(MS_POR_FRAME - time);

                                        }
                                        catch (Exception ex) {}
                                }

                        }
         }

                        sonidosStop();
                        sonidos[Pantalla.SON_EXPLOSION].reproducir(1);
                        sonidoOcupado=true;


                //Para que veamos bien la explosión de nuestra nave
                for (int i=0;i < 50;i++)
                {
                        jugador.anim_a[jugador.animActual].adelante();
                        repaint();
                        serviceRepaints();
                }
                App.gameOver();


}
```

Figura 5.8. Ejecución de la aplicación

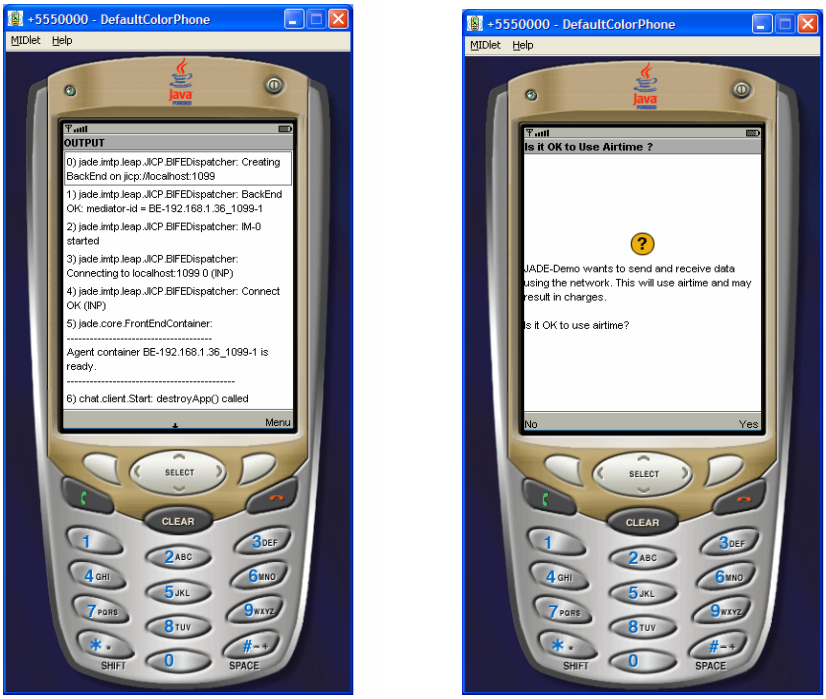

Figura 5.9. Solicitud de acceso a red

Figura 5.10. Registro del jugador

Figura 5.11. Visualización de escena de juego

Figura 5.12. Visualización de escena de juego

6. CONCLUSIONES

Los sistemas de telefonía móvil se caracterizan por una serie de restricciones: ancho de banda limitado, alta probabilidad de error en el interfaz radio, cobertura discontinua y limitada, baja capacidad de procesamiento en los sistemas finales, interfaz de usuario limitada, etc.

La utilización de la tecnología de agentes en estos sistemas y la aparición de versiones de Java para pequeños dispositivos, como j2ME para PDA's o teléfonos móviles -y considerando que el lenguaje de programación en el que más plataformas de agentes móviles se han desarrollado es Java-, hacen posible adaptarse a estas limitaciones para proporcionar mejores servicios a los usuarios finales y mejorar las prestaciones de las aplicaciones en red, porque:

- ✓ Los agentes que proporcionan un servicio pueden enviarse dinámicamente y bajo demanda a los propios usuarios.

- ✓ Los agentes permiten realizar distribución de tareas para realizar actividades de gestión, siendo los propios agentes quienes recopilen datos y los procesen localmente en la parte del terminal móvil.

- ✓ La autonomía de los agentes permite que se realicen tareas de forma asíncrona.

- ✓ Los agentes pueden realizar gran parte del procesamiento de forma local, por lo que se conseguirá una reducción importante del tráfico en la red.

- ✓ Los agentes permiten una mayor independencia de la disponibilidad de la red, ya que su capacidad de movilidad les permite migrar a otros nodos de la red.

En este trabajo he integrado la tecnología de Agentes Móviles en el mundo de los juegos para dispositivos móviles, logrando conseguir una arquitectura que facilite el desarrollo de aplicaciones lúdicas multijugador con agentes móviles personalizados. Para ello, se ha utilizado como plataforma base de desarrollo de Agentes, la plataforma JADE-LEAP [LEAP06].

En concreto, esta propuesta consta de una arquitectura basada en agentes móviles de la que forman parte los agentes Gestor, ControladorPartida, Calculador, VisualizadorJugador y Jugador. Los cuatro primeros funcionan en el Dispositivo Servidor y el último en el Dispositivo del Usuario. El detalle de esta propuesta se analiza en el capítulo 5.

Para evaluar la propuesta, he desarrollado un caso práctico de aplicación lúdica (Star War), siguiendo la arquitectura de agentes planteada. Para la fase de análisis y diseño de la aplicación, sigo la metodología de desarrollo de sistemas Multiagentes INGENIAS [GRASIA05], que sigue un proceso iterativo, incremental y basado en componentes.

Sin embargo, existen un conjunto de problemas abiertos para la implantación de esta tecnología. Por una parte el tema de la seguridad, que es un tema crítico en todas las implementaciones de sistemas de agentes móviles y más aún en el caso de sistemas de telefonía móvil, en los que se debe garantizar itinerancias de servicios y por lo tanto de agentes, a través de diferentes redes de diferentes operadores. El tema es crucial y para su implementación se deberían establecer políticas de seguridad entre organismos de diferentes países.

Por otro lado, la inversión tecnológica de la industria del juego cada vez en aumento, acompañado de la evolución contínua en el rendimiento y prestaciones de los teléfonos móviles, reducen sustancialmente las restricciones para el desarrollo de juegos multijugador en tiempo real de alto nivel. Este hecho está originando la creación de nuevos modelos de negocio ofreciendo más servicios

que ayudados de la tecnología de agentes móviles sugieren un futuro prometedor en este ámbito de estudio.

El desarrollo del presente trabajo constituye una experiencia interesante en la comprensión de la situación actual de las tecnologías de agentes móviles, en particular en el desarrollo de sistemas multiagentes para dispositivos móviles aplicado al escenario de los videojuegos en terminales de capacidad reducida.

6.1 Líneas de Investigación Futuras

En este trabajo han habido varios puntos donde no se ha entrado en detalle, debido a su extensión, y donde se podría investigar más para cubrir las necesidades de proyectos con una complejidad mayor que la que he tenido en cuenta para la definición de la arquitectura propuesta. Esos puntos de extensión podrían pasar por los siguientes:

- ✓ Considerar una optimización de la arquitectura planteada de tal forma que soporte el desarrollo el desarrollo de juegos en línea multijugador de forma masiva, considerando por ejemplo una aplicación distribuida del ámbito de los millares de dispositivos. Una arquitectura apropiada soportada en una plataforma versátil de agentes como JADE-LEAP, podría lograr la puesta en práctica de este tipo de aplicativos de manera simple y rápida.

- ✓ La realización del VHE (Virtual Home Environment), que permitiría la personalización y portabilidad de los servicios de los usuarios independientemente de la red que le da servicio y del terminal que empleen en el acceso. Asociando la implementación del VHE con un agente móvil, que permita configurar el servicio para adaptarse a las preferencias del usuario y a las características del terminal, y además sería el encargado de crear el propio perfil de usuario analizando su comportamiento y su posición.

- ✓ La Portabilidad (los agentes móviles deben ser capaces de moverse en redes heterogéneas), Interoperabilidad (Es necesario que los agentes de un sistema puedan interactuar e intercambiar información con los de otro sistema o bien con objetos de otras tecnologías) e implementación de la tecnología de agentes:
    - Interoperabilidad con otros sistemas de agentes móviles
    - Integración de los agentes móviles en los navegadores
    - Integración de la tecnología con otras tecnologías como CORBA

En cuanto a la seguridad de las plataformas de agentes, se plantean las siguientes líneas de investigación:

- ✓ Mecanismo de distribución de claves públicas entre servidores
- ✓ Protección total de los datos del agente y ejecución privada
- ✓ Restricción del consumo de HD y RAM por agente
- ✓ Intercambio de mensajes e información entre agentes de forma segura

ANEXO

6.1 Bibliografía y Referencias

[AOSE 00] Ciancarini, P. and Wooldridge, M., 2001. Agent-Oriented Software Engineering. First International Workshop AOSE 2000, Lecture Notes in Computer Science Vol. 1957. Springer-Verlag, Berlin.

[BUR96] B. Burmeister. Models and methodology for agent-oriented analysis and design. (1996).

[CARLETON01] Qusay H. Manmoud. MobiAgent: A Mobile Agent-based Approach to Wireless Information Systems. In *Second International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2000)*, 2001.

[CKADS97] Iglesias, C.A.M., Gonzalez, J.C. and Velasco, J.R. 1997. *Analysis and design of multiagent system using MASCommonKADS.* In AAAI 97 Workshop on Agent Theories, Architectures and Languages, Providence, RI, ATAL.

[CONCOR98] Concordia: An Infrastructure for Collaborating Mobile Agents Mitsubishi Electric Research Laboratories (Cambridge, Massachusetts)
http://www.meitca.com

[CELESTE04] Agentes Móviles en computación ubicua
Departamento de Ingeniería Telemática. Universidad Carlos III de Madrid.
Dra. Mª Celeste Campo Vázquez

[CORCHADO02] Juan M. Corchado y Jose M. Molina. En “Introducción a la Teoría de Agentes y Sistemas Multiagente”, Salamanca, España (2002). Edite Publicaciones Científicas.

[DMAT01] DeLoach, S. A. and Wood, M., 2001. Developing multiagent systems with agenttool. In Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop (ATAL 2000), C. Castelfranchi, Y. Lesperance (Eds.). Lecture Notes in Computer Science. Vol. 1986, Springer Verlag, Berlin.

[FIPA02] FIPA. *FIPA Agent Management Specification*, March 2002. http://www.fipa.org

[GARAMENDI04] Agentes Inteligentes: JADE
Juan Fco. Garamendi Bragado. Universidad Rey Juan Carlos. Abril 2004
http://platon.escet.urjc.es:9673/foros/AgentesInteligentes_2003_2004/1082414149/1082495069/Jade.pdf

[GIANP01] Gian Pietro Picco, Mobile agents: an introduction, Microporcessors and Microsystems 25 (2001), 65-74.

[GRASIA05] Grupo de Agentes de Software:Ingeniería y Aplicaciones
Universidad Complutense Madrid
http://grasia.fdi.ucm.es/ingenias/Spain/index.php

[HUHNS97] Readings in Agents (Paperback) by Michael N. Huhns
Publicación: Morgan Kaufmann (October 1, 1997)

[HOLGER97] The Architecture of the Ara Platform for Mobile Agents
Holger Peine and Torsten Stolpmann
Dept. of Computer Science
University of Kaiserslautern, Germany
http://wwwagss.informatik.uni-l.de:80/Projekte/Ara/status.html

[IBM Aglets06] IBM Research Laboratories
http://www.research.ibm.com/trl/aglets/

[IBM05] **Título:** *El negocio de los servicios y aplicaciones de telefonía móvil en el mercado residencial*
Fuente: IBM Business Consulting Services
Fecha: Marzo 2005
http://www-5.ibm.com/services/es/bcs/html/bcs_index.html

[IJCAI99] Jennings, N. R., 1999. Agent-based computing: Promise and perils. In Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2). Stockholm, Sweden, pp. 1429-1436.

[J2ME06] Java™ Platform, Micro Edition (Java ME)
http://java.sun.com/javame/index.jsp

[JADE06] JADE (Java Agent Development Framework)
http://jade.tilab.com/

[JAMES94] James E. White. Telescript technology: Scenes from the electronic marketplace. General Magic White Paper, General Magic, 1994

[JENNINGS98] N. R. Jennings, K. Sycara, M. Wooldridge. *"A Roadmap of Agent Research and Development"*. Autonomous Agents and Multi-Agent Systems, I, 1.998.

[JENNINGS02] Agent Technology: Foundations, Applications, and Markets by Nicholas R. Jennings (Editor), Michael J. Wooldridge.

Publicación: Springer; 1 edition (September 23, 2002)

[KQLM97] D. Benech, T. Desprats, Y. Raynaud. *"A KQML-CORBA based Architecture for Intelligent Agents Communication in Cooperative Service and Network Management"*.

[LANGE96] Danny B. Lange and Daniel T. Chang. *IBM Aglets Workbench: A White Paper*, 1996. IBM Corporation.

[LANGE99] Lange, D. 1999, 'Seven Good Reasons for Using Mobile Agents', In Communication of ACM, vol. 42, no. 3.

[LEAP06] JADE-LEAP: http://jade.tilab.com/

[LUCK03] Understanding Agent Systems (Springer Series on Agent Technology) by Mark d'Inverno, Michael Luck
Publicación: Springer; 2 edition (December 5, 2003)

[MAS01] DeLoach, S. A., et al, 2001. Multiagent systems engineering. In The International Journal of Software Engineering and Knowledge Engineering, Vol. 11, No. 3.

[MAS04] Agentes de Software y Sistemas multiagente; conceptos, arquitecturas y aplicaciones
Ana Mas . - Madrid : Pearson Educación , [2004]

[MASIF98] *"MASIF-RTF Results"*. Object Management Group, 1.998.

[MOLE97] Joachim Baumann. *Mobility in the Mobile Agent System Mole*. In CaberNet: 3rd Plenary Workshop, 1997. http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/1997-baumann-05-paper.html
http://mole.informatik.uni-stuttgart.de/

[MONASH00] Patrik Mihailescu, Elizabeth A. Kendall, and Yuliang Zheng. Mobile agent platform for mobile devices. In *Poster Paper Collection of the Second International Symposium on Agent Systems and Applications (ASA '00). Fourth International Symposium on Mobile Agents (MA '00)*, sep 2000.

[MONASHEC01] Patrik Patrik Mihailescu and Walter Binder. A mobile agent framework for m-commerce. Technical report, Peninsula School of Network Computing. Monash University. CoCo Software Engineering, 2001.

[MORENO02] Using JADE-LEAP to implement agents in mobile devices
Fuente: Antonio Moreno, Aïda Valls, Alexandre Viejo
Grup de Sistemes Multi-Agent (GruSMA) Departament d'Enginyeria Informàtica i Matemàtiques Escola Tècnica Superior d'Enginyeria (ETSE)
Universitat Rovira i Virgili (URV)
Fecha: 2002
http://jade.tilab.com/papers/EXP/02Moreno.pdf

[NAVA02] Nava, Sandra, Federación de Bibliotecas Digitales utilizando Agentes Móviles,
http://ict2.udlap.mx/people/sandra/prop_formal.html,
Noviembre 15 de 2002.

[NEITSIZE05] Small Screens, Global Vision. Netsize Guide 2006 Edition
Fuente: Netsize Group
Fecha: 9 de febrero de 2006
http://www.netsize.com/?id=5&sid=1

[PEYNAM03] Automated service negotiation between autonomous computational agents / Peyman Faratin; foreword by Nick Jennings and Carles Sierra
Publicación: Bellaterra: Institut d'Investigació en Intel·ligència Artificial, [2003]

[PRIETO04] Prieto Martín, Manuel Jesús
Desarrollo de juegos con J2ME : Java 2 Micro Edition / Manuel Jesús Prieto
Paracuellos del Jarama (Madrid) : RA-MA, [2004]

[PRJADE02] Bellifemine, F., et al, 2002. Jade Programmer's Guide (JADE 3.1). http://sharon.cselt.it/projects/jade/

[RUSEL02] High score! : la historia ilustrada de los videojuegos

Rusel DeMaria, Johnny L. Wilson ; [traductor, José Mª Martín] . - Madrid: McGraw-Hill, Interamericana de España , [2002]

[RUP99] Ivar Jacobson, Grady Booch y James Rumbaugh. “The Unified Software Development Process”. Addison Wesley (1999).

[SAIPE01] Jose A. Barcala, Pedro Cuesta, Alma Gómez, Juan C. González, Francisco J. Rodríguez.
Agentes móviles en SAIPE: Sistema de acceso a Información Personal desde Entornos con conectividad limitada

[SANJUAN06] A. Castillo, O. Sanjuán, Y. Sáez y L. Joyanes. Approaching an object-oriented methodology for building software agents in knowledge management. En “Proceedings of SOCCO 2001” (2001).

[SEGFIPA 98] FIPA 98 Part 10 Version 1.0: Agent Security Management Specification (obsolete).
http://www.fipa.org/repository/obsoletespec.html

[SEGFIPA 00] FIPA Agent Management Specification
http://www.fipa.org/repository.

[SEGOVIA03] Aproximación Metodológica para el desarrollo de Software Orientado a Sistemas Multiagente.
Doctor Francisco Javier Segovia. Junio 2003
FACULTAD DE INFORMATICA UNIVERSIDAD POLITECNICA DE MADRID

[SPACEWAR06] Spacewar! was conceived in 1961 by Stephen Russell, and Wayne Wiitanen. It was first realized on the PDP-1 in 1962 by Stephen Russell, and Martin Graetz, together with Alan Kotok, Steve Piner, and Robert A Saunders.
http://fusionanomaly.net/spacewar.html

[TELESCRIPT95] Telescript Technology: The Foundation of the Electronic Marketplace. General Magic Inc., 1995

[UMTS00] Josef F. Huber, Dirk Weiler, Hermann Brand, UMTS, the Mobile Multimedia Vision for IMT-2000: A Focus on Standardization, IEEE Communications Magazine, September 2000.

[VOYAG98] Voyager. “Manual de Referencia de la Plataforma Voyager”. Recursion Software, Inc, http://www.recursionsw.com/mobile-agents.htm

[W3C06] W3C World Wide Web Consortium
http://www.w3.org/XML/9708/ADT

6.2 Glosario

Agente

Es una entidad software que combina uno o más servicios o capacidades en un modelo de ejecución integrado y autónomo. Es el elemento básico de un sistema multiagente.

Agent Communication Language (ACL)

Un lenguaje con una sintaxis, semántica y pragmática formalmente definidas. Es la base de las comunicaciones entre agentes de una naturaleza hetereogénea.

AUML

Una extensión libre de UML para el modelado de agentes

Common Object Request Broker Architecture (CORBA)

Especificación de la OMG para la construcción de plataformas distribuidas independientes del lenguaje de programación.

GPRS (General Packet Radio Service)

Es una tecnología digital de telefonía móvil. Es considerada la generación 2.5, entre la segunda generación (GSM) y la tercera (UMTS). Proporciona altas velocidades de transferencia de datos (especialmente útil para conectar a Internet) y se utiliza en las redes GSM.

GPRS es sólo una modificación de la forma de transmitir datos en una red GSM, pasando de la conmutación de circuitos en GSM (donde el circuito está permanentemente reservado mientras dure la comunicación aunque no se envíe información en un momento dado) a la conmutación de paquetes.

IDL (Interface Definition Language)

Es un lenguaje de especificación de interfaces que se usa como parte de la tecnología CORBA. Ofrece la sintaxis necesaria para definir los métodos que queremos invocar remotamente. Una vez tengamos esta interfaz creada deberemos pasarla por un compilador de interfaces que generará el proxy o stub cliente y el skeleton o stub servidor.

Object Management Architecture (OMA)

Arquitectura distribuida y orientada a objetos utilizada en CORBA.

ORB (*Object Request Broker)*

En Computación distribuida es el nombre que recibe una capa de software (también llamada middleware) que permite a los objetos realizar llamadas a métodos situados en máquinas remotas, a través de una red. Maneja la transferencia de estructuras de datos, de manera que sean compatibles entre los dos objetos.

RMI - Remote Method Invocation

Mecanismo de invocación remota de Java. Al ser RMI parte estándar del entorno de ejecución Java usarlo provee un mecanismo simple en una aplicación distribuida que solamente necesita comunicar servidores codificados para Java. Si se requiere comunicarse con otras tecnologías debe usarse CORBA o SOAP en lugar de RMI.

RPC (Remote Procedure Call)

Protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

RUP (Rational Unified Process)

El Proceso Racional Unificado, es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar

más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

Sistema de Agentes (SA)

Un sistema de agentes o plataforma de agentes, proporciona la infraestructura necesaria para que los agentes puedan se utilizados. Un agente debe ser registrado en una plataforma para poder interactuar con otros agentes de esa plataforma. Un SA contiene tres componentes básicos: ACC, AMS y DF.

Sockets

Concepto abstracto por el cual dos programas (situados en computadoras distintas) pueden intercambiarse cualquier flujo de datos, generalmente de manera fiable y ordenada.

Tcl/tk (Tool Command Language/tk Toolkit)

Es un lenguaje interpretado de scripts de alto nivel desarrollado por Sun Microsystems en 1987

UML (*Unified Modeling Language*)

Lenguaje Unificado de Modelado (UML, es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está apoyado en gran manera por el OMG (Object Management Group).

UMTS (Universal Mobile Telecommunications System)

Es el sistema de telecomunicaciones móviles de tercera generación, que evoluciona desde GSM pasando por GPRS.