TAgentsP y PDP: propuestas para una plataforma de agentes en computación ubicua

Mª Celeste Campo Vázquez, Carlos García Rubio, Florina Almenares Mendoza
Andrés Marín López, Carlos Delgado Kloos
Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid
Avda. Universidad 30. 28911 Leganés, Madrid, España.
e-mail: (celeste, cgr, florina, amarin, cdk)@it.uc3m.es

Resumen— **La computación ubicua según la visión de Weiser, presenta entornos saturados de dispositivos con capacidades de computación y comunicación que le proporcionan a los usuarios servicios ocultando la complejidad tecnológica subyacente. Nuestra línea de investigación pretende potenciar el uso de la tecnología de agentes móviles en computación ubicua para facilitar este objetivo. El trabajo que aquí presentamos se centra por un lado en proporcionar una plataforma de agentes para dispositivos tipo PDA o teléfono móvil, es decir, con capacidad de cómputo y comunicación limitada, integrada dentro de Java 2 Platform Micro Edition (J2ME). La novedad de este desarrollo es permitir la movilidad directa de los agentes entre dispositivos limitados. Por otro lado, uno de los grandes retos que presenta el uso de agentes en computación ubicua es permitir a los dispositivos compartir de manera espontánea entre ellos servicios. En este artículo también presentamos un modelo basado en el uso de un *Directory Facilitator* distribuido, que hace uso de un protocolo de descubrimiento de servicios en computación ubicua, que hemos bautizado PDP.**

I. Introducción

Si nos fijamos a nuestro alrededor, la visión futurista que Mark Weiser describió en su artículo "The Computer for the 21st Century" en 1991 [1] comienza a ser una realidad. Weiser describe entornos saturados de elementos con capacidades de cómputo y comunicación, totalmente integrados en nuestras vidas y que nos proporcionan información asociada a nuestras necesidades y al entorno en el que nos encontrabamos en cada momento. En la actualidad, gracias a la evolución de la tecnología hardware, existen un mayor número de dispositivos: teléfonos móviles, PDAs, pagers, que nos acompañan en todo momento debido a su reducido tamaño. Estos dispositivos tienen capacidad de cómputo y además pueden comunicarse con otros elementos sin necesidad de conexiones físicas, gracias al desarrollo y evolución de los protocolos inalámbricos, tanto en redes celulares, GPRS y UMTS, como en redes locales, WLAN y Bluetooth. Así, ya es más o menos habitual que estos dispositivos nos proporcionen nuevas aplicaciones además de las que propiamente tenían asociadas, por ejemplo, desde un móvil no sólo mantenemos una conversación telefónica, sino que también podemos ver la información del tiempo, localizar la farmacia más cercana o incluso, algo menos común, pero posible, programar la lavadora [2].

Aún así, para alcanzar la visión de Weiser nos queda un importante camino por recorrer, es necesario que estos pequeños dispositivos se integren con los grandes sistemas de computación, para poder utilizar sus servicios y ofrecerselos a los usuarios de manera transparente, independientemente del lugar en el que se encuentra, del dispositivo que utiliza y del tipo de red y protocolo de comunicación con el que accede a ellos. El objetivo es conseguir, lo que en los sistemas móviles de tercera generación se ha denominado acertadamente Virtual Home Enviroment (VHE).

En este escenario, las aplicaciones software de estos pequeños dispositivos deben de adaptarse a las restricciones de memoria y procesamiento que impone el dispositivo en el que se ejecutan, y a una comunicación intermitente y de calidad cambiante, necesitan autonomía propia para poder alcanzar los objetivos que quiere el usuario, pero sin necesidad de interaccionar continuamente con él, y deben ser capaces de moverse por otros sistemas para poder obtener información o ejecutar tareas que las limitaciones del dispositivo no les permiten realizar de forma local. El paradigma de computación que se adapta a todas estas características es lo que se denomina Agente.

Éste es el motivo que nos ha llevado a centrar nuestro trabajo actual en aplicar la tecnología de agentes a entornos de computación ubicua. Aunque esta tecnología se adapta a este tipo de entornos y ya existen muchas implementaciones, estudios y estándares al respecto, no se habían involucrado a pequeños dispositivos, por lo que es necesario realizar un análisis detallado de todos los aspectos desarrollados hasta este momento para adaptarlos a los nuevos requisitos y retos que se nos plantean.

Hemos abordado la realización de un entorno de ejecución de agentes móviles en dispositivos limitados. Las contribuciones que presentamos en este artículo pueden dividirse claramente en dos partes.

- Por un lado, debido a que la mayoría de plataformas de agentes móviles se han desarrollado en Java y a la reciente aparición de Java para pequeños dispositivos, nuestro análisis se centrará en plataformas de agentes desarrolladas en Java. En la sección II, se hará una breve revisión de las plataformas de agentes y en concreto, de las desarrolladas en Java para entorno PC. Después se analizará, sección III, la versión de Java para dispositivos con capacidades limitadas, J2ME (Java 2 Micro Edition) para detectar que características del lenguaje Java, que lo hacían un buen lenguaje para desarrollar plataformas de agentes, no se han mantenido en esta versión y por lo tanto, que problemas deberán solucionarse. Se describirán brevemente en esta sección también las propuestas existentes en la litera-

tura sobre plataformas de agentes empleando J2ME. En la sección IV, describiremos detalladamente nuestra propuesta y el estado de nuestros desarrollos actuales.

- Por otro lado, uno de los grandes retos que presenta el uso de agentes en computación ubicua es permitr a los dispositivos compartir entre ellos servicios. En la sección V presentamos el problema del descubrimiento de servicios y los principales protocolos de descubrimiento de servicios propuestos hasta el momento. Veremos que ninguno de ellos se adapta bien al caso de la computación ubicua. En la sección VI proponemos la distribución y fragmentación de la función de DF del modelo MAS de FIPA entre los dispositivos que forman una red de computación ubicua. En la sección VII proponemos un protocolo de descubrimiento de servicios adaptado a la computación ubicua, el Pervasive Discovery Protocol, PDP.

II. Tecnología de agentes

La tecnología de agentes ha tenido un especial impacto en los últimos años gracias a su aplicación en la computación distribuida. El modelo más ampliamente extendido es el modelo cliente/servidor, en el que un cliente que se ejecuta en un entorno envía un conjunto de datos a un servidor, y espera que éste le envíe los datos de respuesta de la operación realizada, antes de enviar nuevos datos. Cada mensaje intercambiado en la red implica una petición de un servicio y una respuesta a esa petición. La comunicación establecida precisa de una conexión permanente. Esto provoca el consumo de un gran número de recursos.

La introducción de la tecnología de agentes permite realizar las mismas operaciones pero con la ventaja de que sean asíncronas y que además no precisemos 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.

A pesar de estos beneficios, debemos tener en mente que la tecnología de agentes no sustituye a otros paradigmas de la computación distribuida, una aplicación que se desarrolla con tecnología de agentes puede desarrollarse con las tecnologías convencionales. Lo que debemos analizar es si, para una aplicación determinada, con la tecnología de agentes se obtienen mejores prestaciones, y en ese caso aplicarla. Existen algunos campos de aplicación en los que se ha demostrado que la aplicación de agentes es mucho más beneficiosa: en concreto se han realizado estudios en el caso de acceso a bases de datos [3] y en tareas de gestión de red [4].

Existen dos conceptos fundamentales en los sistemas de agentes móviles, que son el concepto de agente y el de plataforma de agente.

A. Agentes móviles

Un agente se define como una entidad software que actúa en nombre de otra entidad, por ejemplo, de una persona o de otro agente. Que es autónomo y se comporta según los objetivos que debe de alcanzar. Reacciona ante eventos externos y puede comunicarse y colaborar con otros agentes.

Un agente móvil es un agente que puede migrar entre dos nodos de una red. Junto a este tipo de agentes, surgieron también los agentes inteligentes, basados en aplicar conceptos de inteligencia artificial al paradigma de agentes. Un agente móvil puede ser inteligente y a la inversa, pero ambas características son independientes.

Para aclarar la definición de agente móvil es necesario indicar que es lo qué entendemos por migración de un agente entre dos nodos. Una entidad software en ejecución está compuesta por el código, que nos proporciona la descripción estática de su comportamiento, y su estado, que nos proporciona su descripción en un momento determinado de ejecución.

El estado contiene por una parte, lo que se denomina espacio de datos, que son los recursos accesibles desde todas las rutinas activas, y por otra, la información de control, compuesta por valores como el del contador de programa, estado de la pila, etc... Migrar un agente consiste en enviar su código y el estado de ejecución al host remoto, de forma que después de su migración en el host remoto se retome su ejecución en el mismo punto en el que se encontraba antes de migrar.

La complejidad que presenta la migración del estado de un código en ejecución se ha abordado de diferentes formas, en concreto se pueden diferenciar dos grandes grupos, los que soportan:

- Movilidad fuerte (strong mobility): se migra el código y estado, abarcando tanto el espacio de datos como el estado de ejecución.
- Movilidad débil (weak mobility): se migra el código y el espacio de datos del estado del código, pero no el estado de ejecución.

B. Plataformas de agentes móviles

Los agentes móviles requieren un entorno especial para su ejecución. Este entorno es lo que se denomina también plataforma. La plataforma además de aportar la capacidad de ejecución propiamente dicha, debe de proporcionar una serie de servicios básicos:

- Movilidad: que facilite la migración de agentes entre nodos remotos.
- Comunicaciones: que facilite la comunicación de los agentes con el mundo exterior, principalmente, que le permita interactuar con otros agentes en tareas cooperativas.
- Nombrado: que permite nombrar a los agentes de manera que puedan ser indentificados de forma unívoca, y también nombrar a las plataformas y nodos a los que migran los agentes.
- Descubrimiento y localización: que facilite a los agentes descubrir otros agentes y plataformas con los que puede interactuar, o las que pueda migrar, para alcanzar los objetivos que tiene encomendados.
- Seguridad: que garantice por una parte la protección de los agentes ante ataques del host en el que se ejecuta y por otra, la protección del host ante los ataques de los agentes que se ejecuten en él.

Existen diversas plataformas de agentes móviles entre las que cabe destacar Aglets de IBM, Voyager de ObjectSpace

y Grasshopper de IKV, las tres hechas en Java.

C. Beneficios de los agentes móviles

Tradicionalmente, los agentes móviles se han contemplado como una alternativa viable en la que la transmisión del código necesario para realizar una tarea resultaba más económico (en términos del tráfico necesario) que la orientación cliente / servidor. En este sentido en [5] se describe una comparativa de sincronizaciones de agendas entre máquinas remotas en tres casos: el primero basado en cliente / servidor, el segundo basado en optimizar el servidor para la operación de sincronización, y el tercero basado en agentes móviles. Las conclusiones del trabajo fueron que la alternativa de cliente / servidor crecía linealmente con el número de participantes, mientras que las otras permanecían casi constantes y a un nivel muy inferior (en términos de datos transmitidos). Hay que hacer notar que el escenario utilizaba solo dos servidores que almacenaban las agendas. La segunda conclusión era que el coste de desarrollar el servidor optimizado fue bastante grande, desarrollo que no fue necesario en el caso de agentes móviles.

Además del menor consumo de recursos de comunicación y del menor coste de desarrollo de aplicaciones concretas, el paradigma de agentes y sistemas multiagentes plantea otros beneficios adicionales, como son la cooperación entre agentes y el diseño de sistemas dirigidos al usuario, en los que el agente personal del usuario cobra una gran importancia. Como última ventaja citaremos la gran diversidad de dispositivos a disposición del usuario, que los agentes móviles pueden integrar para facilitar sus tareas y dar una visión de servicios ubicuos, en lugar de servicios diversos prestados por distinto software corriendo en dispositivos ubicuos.

D. Lenguajes de programación de plataformas de agentes

Una parte importante del trabajo realizado en el desarrollo de plataformas de agentes ha sido proporcionar o adaptar lenguajes de programación para implementar estas plataformas [6]. Se han utilizado lenguajes de propósito más general como pueden ser Java o adaptaciones de otros, como las versiones realizadas sobre Tcl: Safe-Tcl, Agent Tcl, TACOMA y en algunos casos, se han definido lenguajes específicos para este propósito: Telescript, M0, Tycoon.

La característica principal que deben proporcionar estos lenguajes es permitir la movilidad de una unidad de ejecución, que consiste en una parte de código, que nos proporciona la descripción estática del comportamiento de un programa, y el estado de la unidad de ejecución. El estado contiene por una parte, lo que se denomina espacio de datos, que son todos los recursos accesibles desde todas las rutinas activas y por otra, lo que se denomina estado de ejecución, que es información de control, como el valor del contador del programa y el estado de la pila, que nos permite retomar la ejecución después de la migración. Cada uno de estos componentes pueden moverse independientemente.

Los diferentes lenguajes han aportado soluciones diferentes para la movilidad de estos componentes, agrupándose en dos clases, por una parte los lenguajes que soportan *strong mobility* y los que se soporta *weak mobility*:

- *Strong mobility*: tienen la capacidad de migrar código y estado, abarcando tanto el espacio de datos como el estado de ejecución.
- *Weak mobility*: tienen la capacidad de migrar código y el espacio de datos del estado del código, pero no el estado de ejecución.

Aunque existen lenguajes que proporcionn *Strong mobility*, como son Telescript, TACOMA, Ara y DÁgents. La mayoría de los lenguajes soportan *weak mobility*, entre ellos Java, que a pesar de esta limitación ha sido el lenguaje en el que más plataformas de agentes se han desarrollado, debido a las siguientes ventajas aportadas por el lenguaje:

- Independencia de la plataforma: una de las principales ventajas que introdujo Java, fue poder desarrollar software independiente del procesador. La clave consistió en desarrollar un código neutro que pudiera ser ejecutado sobre una máquina virtual, denominada Java Virtual Machine, que es la que interpreta este código neutro convirtiéndolo a código particular de cada CPU o plataforma. Esto nos permite, que cualquier agente desarrollado en Java pueda ejecutarse en un host remoto que tenga una máquina virtual Java.
- Ejecución segura: Java fue concebido para su utilización en redes como Internet, por lo que la seguridad ha adquirido una importancia vital en su definición. La arquitectura de seguridad de Java hace que sea relativamente sencillo salvaguardar a un host de un agente malicioso, porque no se permite el acceso directo a los recursos del host, es un lenguaje fuertemente tipado y no existen punteros.
- Carga dinámica de clases: este mecanismo permite a la máquina virtual cargar clases en tiempo de ejecución, e incluso se puede habilitar su descarga a través de la red. Esto facilita la migración de agentes de un host a otro, ya que sólo es necesario que migre la clase root del agente, las demás clases que se precisen en ejecución, y que no existan en el host destino, serán descargadas dinámicamente, bien desde el host origen o bien desde un repositorio de clases habilitado en la plataforma de agentes para proporcionar este servicio.
- Programación multithread: los agentes por definición son autónomos, es decir, un agente se ejecuta independientemente de los otros agentes que residen en el mismo lugar. El comportamiento autónomo de los agentes se consigue permitiendo que cada agente se ejecute en su propio proceso ligero, también llamado hilo de ejecución. Java permite la programación multihilo y además proporciona un conjunto de primitivas de sincronización que nos permite la interacción entre los agentes.
- Serialización de objetos: el lenguaje Java proporciona mecanismos de serialización que permiten representar el estado de un objeto en forma serializada con el suficiente detalle para que este objeto se pueda reconstruir posteriormente.
- Reflexión: El código Java permite obtener información sobre los campos, métodos, y constructores de la clases cargadas. Ésta información puede emplearse para crear ob-

jetos de una clase que se descubre en tiempo de ejecución. Esto permite dotar de cierta inteligencia a los agentes ya que pueden obtener información de si mismos y de otros agentes.

A pesar de todas estas ventajas, el desarrollo de plataformas de agentes con Java tiene ciertas limitaciones, una de las más importante es la que comentamos anteriormente, que sólo soporta *weak mobility*, pero además:

- No proporciona un soporte adecuado para el control de recursos, ya que en Java no se tiene control sobre los recursos consumidos, tiempo de procesador y memoria consumida, por un objeto.
- No proporciona control de referencias, todos los métodos públicos de un objeto Java son accesibles desde cualquier otro objeto que tenga una referencia a él. Esto presenta un problema ya que los agentes no tienen control sobre qué agentes acceden a sus métodos.

III. Plataforma de agentes sobre dispositivos limitados

Con la aparición de versiones de Java para pequeños dispositivos, J2ME para PDAs 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 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.

A. 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.

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.

Una de las principales ventajas de J2ME es que es una arquitectura modular, figura 1, que se adapta a las limitaciones de los diferentes dispositivos en las que se quiere integrar. Se definen tres capas:

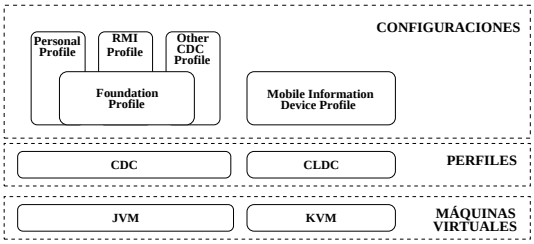

Fig. 1. Arquitectura J2ME

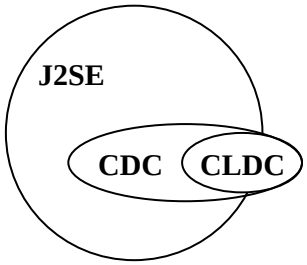

Fig. 2. Relación entre CLDC y CDC y la J2SE

- **Máquina virtual**. En la actualidad J2ME soporta dos máquinas virtuales: la Java Virtual Machine 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. La KVM (K-kilo) es una nueva implementación de una máquina virtual Java, que se ha optimizado en tamaño para que se pueda integrar en dispositivos limitados en memoria, pero que es capaz de aceptar el mismo formato de class-files que la clásica JVM. En el futuro se contempla la posibilidad de aumentar el número de máquinas virtuales soportadas por la plataforma.
- **Configuraciones**. Definen 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.

  Ambas configuraciones tiene clases comunes con la J2SE, que permite la compatibilidad, pero poseen además clases específicas para los tipos de dispositivos para los que se definieron, ver figura 2.
- **Perfiles**. Definen un conjunto de APIs 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.

Nuestro trabajo se centra en los dispositivos que se engloban en la configuración CLDC, por lo tanto nos centramos en el análisis de las limitaciones que presenta esta plataforma:

- Limitaciones del lenguaje Java:
  - No soporta tipos de datos de coma flotante (float o double).
  - No soporta la finalización de instancias de clases. El método Object.finalize() no existe.
  - Limitaciones en el manejo de errores. La mayoría de las subclases de java.lang.Error no están soportadas y en general, la gestión de errores es particular de la implementación realizada para el dispositivo concreto.
- Limitaciones de la máquina virtual:
  - No soporta Java Native Interface (JNI), debido a:
    - Modelo de seguridad limitado del CLDC.
    - Limitaciones de memoria necesarias para el soporte completo de JNI.
  - No soporta cargadores de clase definidos por el usuario, debido a restricciones de seguridad.
  - 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. El modelo soportado por J2ME/CLDC es de tipo "sandbox", es decir las aplicaciones se ejecutan sobre un entorno limitado en el que la aplicación solo puede acceder a las APIs definidas por el CLDC y por los perfiles proporcionados o a clases específicas del dispositivo.

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

Si recordamos las características que convertían a Java en un buen lenguaje para desarrollar plataformas de agentes, sección II-D, 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).

B. Tecnología de agentes y J2ME

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

B.1 LEAP

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 móviles conforme al estándar FIPA (Foundation for Intelligent Physical Agents) que pueda operar tanto en dispositivos móviles (teléfonos móviles, PDA, pagers) como en PCs. El proyecto comenzó en Enero de 2000 y tiene dos fases, la primera de ellas, ya finalizada, consistió en la revisión de los estándares FIPA y WAP y el diseño e implementación de una plataforma de agentes para dispositivos móviles. La segunda fase consiste en evaluar la plataforma en sistemas reales y analizar las prestaciones obtenidas en dos aplicaciones: asistencia en carretera y en tareas de gestión de red.

Para que la plataforma de agentes desarrollada en este proyecto opere tanto en sistemas PCs como en dispositivos móviles con capacidades limitadas, han diseñado una arquitectura modular estructurada en dos partes:

- Parte obligatoria, compuesta por varios módulos, uno denominado kernel independiente del dispositivo y otros dedicados a las comunicaciones y dependientes del dispositivo en el que se ejecute la plataforma.
- Parte opcional, compuesta por varios módulos para interfaces gráficas, parsers, modelado de usuario y datos.

La plataforma de agentes se configurará así mediante un instalador que compondrá los módulos necesarios para el dispositivo concreto en el que se instale.

La plataforma de agentes está desarrollada en Java, consiguiendo de esta manera la independencia del sistema operativo. Para dispositivo móviles emplean las extensión correspondiente de Java, la J2ME/CLDC y para entornos PCs la J2SE. El kernel de la plataforma solamente emplea las APIs comunes a ambas especificaciones de Java.

En el desarrollo de la plataforma de agentes LEAP se han reautilizado dos piezas de sofware desarrolladas por algunos de los miembros del consorcio: por una parte JADE (Java Agent DEvelopement Framework), que es una plataforma de agentes desarrollada por CSELT (Centro Studi E Laboratori Telecommincazioni) conforme a la especificación FIPA97 y ZEUS, que es un toolkit gráfico de desarrollo de agentes diseñado por BT y adaptado en este caso para la creación y gestión de agentes JADE.

B.2 Monash University

En Monash University,[7] [8] [9], 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 la plataforma de agentes, obteniendo de esta forma unas mejores prestaciones, pero obligando a que los dispositivos tengan esta KVM reconstruida.

B.3 School of Computer Science Carleton University

En School of Computer Science Carleton University,[10], también se están realizando desarrollos orientados a la utilización de tecnología de agentes para aplicaciones en entor-

nos 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.

IV. TAgentsP: nuestro perfil sobre CLDC para agentes móviles

Como hemos visto en el apartado anterior algunas de las iniciativas que existen en la literatura simplemente se reducen a permitir que un agente se ejecute en un dispositivo, empleando directamente el entorno de ejecución de J2ME o incluso, algunas más ambiciosas y completas como LEAP, aunque diseñan una arquitectura lo suficientemente generica para posibilitar movilidad de agentes entre dispositivos, no se enfrentan al problema real de implementarlo, y en sus pilotos, simplemente envían agentes a los dispositivos que pueden comunicarse con otros agentes para alcanzar sus objetivos, pero en ningún momento se mueven a otros sistemas para alcanzarlos.

A. Motivación

En nuestro trabajo nos proponemos el reto de implementar una plataforma de agentes en dispositivos limitados CLDC-J2ME que permita que los agentes se puedan mover entre estos dispositivos, sin necesidad de involucrar a dispositivos no limitados, para realizar sus tareas.

Nuestros desarrollos actuales se centran en complementar el MIDP para construir un perfil sobre el CLDC que proporcione la funcionalidad básica de una plataforma de agentes móviles, a este perfil lo hemos llamado TAgentsP (Travel Agents Profile). Cuando hablamos de funcionalidad básica nos referimos a dotar a J2ME de aquellas características que tenía Java como lenguaje de desarrollo de plataformas de agentes y que J2ME no posee, en concreto las que nos permiten *weak mobility*:

- Carga dinámica de clases.
- Serialización de objetos.

Una vez proporcionados los servicios básicos, y para conseguir nuestro objetivo de que los agentes se puedan mover entre dispositivos limitados, necesitamos permitir que se proporcionen unos a otros las clases que precisan los agentes para su ejecución, para ello vamos a implementar un reducido servidor HTTP en cada dispositivo. Esta implementación también nos permitirá la comunicación a nivel agente.

En la figura 3 vemos la arquitectura de nuestra propuesta y en las siguientes secciones describiremos cada uno de los módulos que estamos desarrollando.

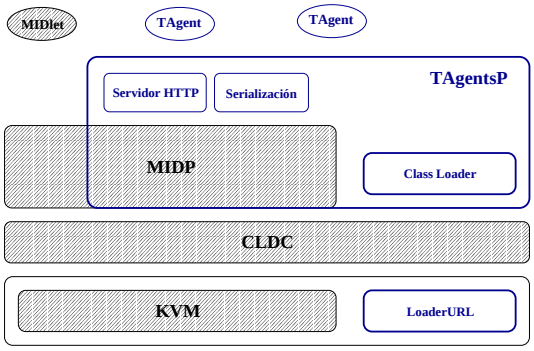

Fig. 3. Arquitectura de nuestra propuesta

B. Carga dinámica de clases en KVM

B.1 ¿Para qué precisamos carga dinámica de clases?

Cuando un agente migra a otro dispositivo, la máquina virtual en el que se va a ejecutar debe de cargar nuevos ficheros de clases para que pueda invocar los métodos del nuevo objeto, todo esto en tiempo de ejecución, para que no sea necesario rearrancar la máquina virtual y por lo tanto, detener todos los procesos en curso. Como hemos visto, CLDC no soporta este tipo de carga dinámica de clases, aunque la máquina virtual sobre la que se implementa, la denominada KVM, no precisa ser rearrancada para cargar nuevos ficheros de clases, es decir, esta restricción se ha impuesto en la implementación del CLDC, justificándolo por motivos de seguridad, ya que debido a las limitaciones de estos dispositivos no ha sido desarrollado todo el nivel de seguridad que Java, en su versión J2SE, proporciona.

Muchos desarrolladores se han decepcionado al ver esta restricción y en futuras versiones de CLDC permitirán que exista carga dinámica de clases y que el propio desarrollador pueda definir sus cargadores de clases, los ClassLoader de J2SE. Mientras esta nueva versión no está disponible, es necesario tener alguna solución temporal que nos permita obtener esta funcionalidad imprescindible para conseguir movilidad de agentes.

B.2 Soluciones actuales

En la literatura existen algunas soluciones temporales a la carga dinámica de clases, algunas ya obsoletas porque se han implementado sobre versiones anteriores de CLDC, pero que merece la pena comentar debido a las ideas que han sido aportadas en ellas.

B.2.a Dynamic Classloading in the KVM. [11]. Razvan Dragomirescu proporcionó una solución a la carga dinámica de clases, para la versión Beta1 de KVM en PalmOS, su idea era descargarse las clases y almacenarlas en el formato propietario para PalmOS antes de que la KVM buscara estas clases para ejecutar un determinado objeto. Para ello implementó sobre el CLDC una clase llamada ClassLoader. El código que desarrolló es libre.

B.2.b JiniME. [12]. Alan Kaminsky ha definido un perfil sobre CLDC para Jini, el JiniME. En Jini se debe dar soporte a la movilidad de objetos entre sistemas, y por lo tanto, ha tenido que enfrentarse a problemas similares a los nuestros en TAgentsP. Respecto a la carga dinámica de clases, su solución pasa por definir una nueva clase en CLDC Classpath que permite añadir al classpath del sistema URL donde se encuentran los ficheros de clase.

B.3 Nuestra modificación de la KVM

Nuestra solución añade dos módulos provisionales a la arquitectura CLDC/KVM, uno en código C, que representamos en la figura como URL Loading, que modifica la KVM para que pueda cargar ficheros de clases a partir de la URL en la que se encuentran, y otro sobre el CLDC, la clase ClassLoader que nos permite añadir/eliminar del classpath del sistema URLs, siempre añadidas al final, para que no se sobreescriban clases del sistema.

El URL Loading al ser código nativo será necesario migrarlo a las diferentes sistemas operativos que queremos involucrar en nuestro desarrollo, en concreto, se realizará su migración a PalmOS y a Symbian, siguiendo las recomendaciones de SUN para el porting de la KVM.

Recordemos aquí que esta solución es provisional mientras no se proporcione una CLDC/KVM con soporte para carga dinámica de clases.

C. Serialización de objetos sobre CLDC

C.1 ¿Para qué precisamos serialización de objetos?

Solucionando el problema de la carga dinámica de clases, hemos permitido que un agente puede seguir ejecutándose en una plataforma remota, ahora debemos abordar el problema de conservar el estado de ejecución del agente cuando migra a esa nueva plataforma, para ello es necesario poder almacenar el estado del agente como una secuencia de bytes de tal forma que en la nueva plataforma pueda reconstruirse el agente manteniendo su estado.

Como hemos visto CLDC/KVM no proporciona mecanismos de serialización, aunque si mecanismos de almacenamiento persistente sobre el que se pueden construir estos métodos.

C.2 Soluciones actuales

En la literatura sólo hemos encontrado una iniciativa a la serialización de objetos propuesta en el artículo que hemos mencionado anteriormente, JiniME. Consiste en definir un interfaz Moveable, con dos métodos, uno para serializar y otro para deserializar objetos. Su solución es metodológica, consiste en obligar al programador a que todos los objetos que quiera que se muevan de un sistema a otro deben de pertener a clases que implementen el interfaz Moveable, pero la implementación de los métodos de serialización y deserialización deben de ser desarrolladas por cada programador.

C.3 Nuestras propuestas de serialización de objetos sobre CLDC

Cuando hemos abordado la solución de la serialización de objetos en J2ME hemos considerado dos posibilidades, una de ellas es similar a la propuesta en JiniME, en concreto la clase base de nuestros agentes tendrá un método, por defecto, que permita serializar y deserializar un agente, y el programador deberá sobreescribir cuando programe sus propios agentes, de manera que garantice que se puedan reconstruir conservando su estado en la plataforma remota.

Otra de las soluciones que estamos implementando es la generación automática de los métodos de serialización/deserialización para una clase determinada dado su fichero fuente. Para ello partiendo de una gramática de Java generaremos un parser que automáticamente incluya estos métodos en los ficheros fuente.

Ambas soluciones se construyen sobre MIDP, ya que emplean el paquete RMS para el almacenamiento de los agentes serializados de forma persistente.

D. Exportar ficheros de clases

D.1 ¿Para qué precisamos exportar ficheros de clases?

Cuando el agente comienza su ejecución en la plataforma remota deberán proporcionarse además los ficheros de clase que emplee y que no necesariamente están en el sistema al que migra, estos ficheros de clases pueden ser proporcionados por la plataforma origen en la que se encontraba inicialmente el agente o por repositorios especiales dedicados a proporcionar ficheros de clase.

D.2 Soluciones actuales

En JiniME se proporciona una solución a este problema, creando un pequeño servidor HTTP en los dispositivos limitados de manera que sirva los ficheros de clase a la plataforma remota a la que migra el agente.

D.3 Servidor HTTP para proporcionar ficheros de clases entre dispositivos

La solución que hemos desarrollado sigue la misma línea que la solución anterior. Si consideramos que el único requisito de comunicación que deben de poseer los dispositivos limitados MIDP/CLDC es soporte para HTTP y que no queremos que un sistema no limitado sea obligatorio para la movilidad de los agentes entre dispositivos limitados, la solución pasa porque en cada uno de ellos tengamos un servidor HTTP que sea capaz de servir los ficheros de clases.

Esta solución se construye sobre MIDP, complementando el paquete proporcionado para conexiones HTTP, añadiendo un listener que escuche peticiones realizadas a una determinada URL y que responda a estas peticiones proporcionando el fichero de clase proporcionado.

Esta implementación también nos permitirá comunicar agentes entre si, mediante el protocolo HTTP.

V. Descubrimiento de servicios

Nuestra línea de trabajo pretende realizar un estudio y aportar soluciones a la adaptación de la tecnología MAS (Multi Agent Systems) a entornos ubicuos. Un elemento fundamental en MAS es la infraestructura o plataforma de agentes, que permite que los agentes realicen las tareas que tienen encomendadas, proporcionándoles un entorno de ejecución, y conjunto de servicios básicos, que el estándar FIPA engloba en tres componentes:

- Agent Management System (AMS): gestión del ciclo de vida de los agentes, gestión de recursos locales, gestión de los canales de comunicación y un servicio de páginas blancas, que permite localizar agentes por su nombre.

- Directory Facilitator: servicio de páginas amarillas, que permite localizar agentes por sus capacidades y no por su nombre.
- Agent Communication Channel: gestión del envío de mensajes entre agentes de la misma plataforma o de plataformas distintas.

En los apartados anteriores hemos abordado la realización de un entorno de ejecución de agentes móviles sobre J2ME. Ahora nos centramos en proporcionar los servicios básicos, en concreto el descubrimiento de servicios, que trataremos en los apartados siguientes.

El descubrimiento de servicios de manera dinámica no es algo nuevo, y ya existen varias propuestas al respecto, más o menos extendidas en entornos de redes fijas: SLP, Jini, Salutation, SSDP de UPnP, ...

Service Location Protocol (SLP) [13] es el principal resultado del Service Location Protocol Working Group del IETF (SVRLOC). Su objetivo es la definición de un protocolo de descubrimiento automático de servicios en redes IP. La infraestructura SLP consiste en tres tipos de agentes: los Agentes de Usuario (UA), son los que realizan el descubrimiento de servicios para satifacer las necesidades que demandan las aplicaciones de los usuarios finales; los Agentes de Servicio (SA), son los responsables de anunciar las características y localización de los servicios y los Agentes de Directorio (DA), son los responsables de almacenar información sobre los servicios que se están anunciando en la red. SLP tiene dos modos de funcionamiento, con Agente de Directorio, en cuyo caso los DA registran en él los servicios que ofrecen y los UA buscan en él los servicios que precisan; o sin DA, en cuyo caso los UA envían por multicast peticiones de servicios a las que los SA que ofrecen el servicio responden mediante mensajes unicast.

Jini [14] es una arquitectura distribuida orientada a servicios desarrollada por Sun Microsystems. El objetivo general de Jini es convertir a la red en una herramienta flexible y fácilmente administrable en la cual los clientes puedan encontrar servicios de un modo robusto y flexible. Uno de los componentes claves de Jini es el Servicio de Búsqueda Jini (JLS - Jini Lookup Service), el cual mantiene información dinámica acerca de los servicios disponibles en una federación (conjunto de servicios) Jini. El JLS es obligatorio dentro de la arquitectura de Jini, y los clientes y los servicios siempre se descubren a través de él y nunca de forma directa.

Salutation [15] fue desarrollado por un consorcio de compañías, con el objetivo de resolver el problema del descubrimiento y acceso a servicios entre un amplio conjunto de dispositivos y equipos en un entorno cambiante. Salutation es un estándar abierto independiente de sistemas operativos, protocolos de comunicaciones y plataformas hardware. La arquitectura Salutation define una entidad llamada SLM (Salutation Manager) que permite a los clientes descubrir y comunicarse con los diferentes servicios proporcionados en la red. El proceso de descubrimiento de servicios puede realizarse a través de múltiples SLMs. Un SLM puede descubrir otros SLMs remotos y determinar los servicios que están registrados allí. De esta manera el descubrimiento de servicios se realiza de una manera mucho más rápida.

Simple Service Discovery Protocol SSDP [16] está definido dentro de Universal Plug-and-Play UPnP, para el descubrimiento dinámico de servicios en redes IP y es una alternativa clara a SLP. SSDP puede operar con o sin un Servicio de Directorio en la red. SSDP opera sobre los protocolos abiertos existentes, usando HTTP (HyperText Transfer Protocol) tanto sobre unicast como sobre multicast. Cuando un servicio quiere unirse a la red, primero envía un mensaje de anuncio con el fin de notificar al resto de los dispositivos su presencia, este anuncio se puede enviar por multicast, de tal forma que si está presente en la red un Servicio de Directorio, éste puede registrar tales avisos y además otros dispositivos en la red pueden ver directamente estos avisos. También existe la opción de enviar un mensaje unicast directamente al Servicio de Directorio. El mensaje de anuncio contiene una URL que identifica el servicio anunciado y una URL a un archivo que proporciona una descripción del dicho servicio. Cuando un cliente quiere descubrir un servicio, puede tanto contactar con el servicio directamente a través del URL que ha obtenido de algún anuncio del servicio, o puede enviar activamente enviar un mensaje de búsqueda de servicio por multicast. En el caso de descubrir un servicio a través de un mensaje de búsqueda, la respuesta puede ser proporcionada por el propio servicio o por un Servicio de Directorio.

Estas soluciones no pueden ser empleadas directamente en el problema que intentamos abordar, debido a que tenemos:

- Entornos rapidamente cambiantes.
- Protocolos de comunicación inalámbricos.
- Capacidades limitadas, tanto de almacenamiento como de procesamiento, de muchos dispositivos.

La principal motivación para no poder utilizar estas soluciones clásicas es que en todas ellas aparece un servidor central en el que los que ofrecen servicios se registran y al que los clientes preguntan cuando quieren encontrar un determinado servicio. Como en computación ubicua las redes son muy dinámicas y los dispositivos tienen capacidades de almacenamiento limitadas es muy costoso que uno de ellos afronte el papel de servidor.

Partiendo de la necesidad de no tener un servidor central, se nos plantean dos posibilidades:

- Métodos “push”: se producen anuncios continuos de los servicios y los clientes escuchan estas peticiones seleccionando el servicio que les interesa.
- Métodos “pull”: son los clientes los que realizan peticiones continuamente para que los servicios se descubran bajo demanda.

Un factor importante a tener en cuenta en estas soluciones en las que, o los anuncios de los servicios, o las peticiones, se envían a todos los dispositivos que están a su alrededor como mensajes de broadcast, suponen un gran gasto de las baterías de los dispositivos y por lo tanto, tiene que optimizarse y valorar cuál es el mejor método para que el descubrimiento de servicios se realice lo antes posible sin implicar un gran número de transmisiones.

Recordemos además, que en entornos tan dinámicos los servicios estarán disponibles un tiempo limitado, por lo que tenemos el compromiso de implementar mecanismos que permitan detectar lo antes posible tanto la disponibilidad como indisponibilidad de esos servicios.

El DEAPspace group del IBM Research Zurich Lab se encuentra en la actualidad proponiendo soluciones a este problema, partiendo de la misma premisa que nosotros: el protocolo de descubrimiento de servicios en un entorno de computación ubicua no debe involucrar a un servidor central. El DEAPspace Algorithm, [17], se basa en un método "push" puro, en el que todos los dispositivos mantienen un "world view" que transmiten a sus vecinos mediantes mensajes de broadcast y que actualizan escuchando el "world view" que tienen los demás.

VI. Directory Facilitator Distribuido

En este apartado planteamos los problemas que presenta el Directory Facilitator, servicio de páginas amarillas en entornos de computación ubicua.

Para implementar el DF en un entorno de computación ubicua, nosotros proponemos implementar un DF distribuido, formado por el trabajo conjunto de DFs de funcionalidad reducida, los llamados "fragmentos DF", de los dispositivos limitados. Los fragmentos que forman un DF distribuido cambian, a medida que los dispositivos llegan y abandonan la red.

Este servicio de DF distribuido permitirá obtener los servicios que están disponibles, el sistema que proporciona el servicio y los medios a través de los que puede acceder. Aunque no proporcionará la forma de acceder al mismo.

Para hacer su trabajo, un fragmento DB tiene que implementar un protocolo de descubrimiento de servicios. Debido a las deficiencias encontradas en los protocolos existentes de descubrimiento de servicios, los primeros pasos en nuestro trabajo se han enfocado a la definición de un nuevo algoritmo de descubrimiento de servicios adaptado a entornos de computación ubicua, denominado Pervasive Discovery Protocol, PDP.

Un fragmento DF podrá interaccionar con cualquier plataforma de agentes o sistema que implemente PDP como protocolo de descubrimiento de servicios, de tal forma que permitirá a los agentes descubrir servicios proporcionados por otros agentes, por plataformas de agentes o por otro tipo de sistemas en general.

En el siguiente apartado describeremos el algoritmo PDP.

VII. Pervasive Discovery Protocol Algorithm

PDP surgió con el objetivo de definir un algoritmo de descubrimiento de servicios que se adapte a entornos de computación ubicua mejor que las soluciones clásicas, pensadas para el descubrimiento automático de servicios en redes fijas y cuya utilización en estos entornos no es óptima cuando la mayoría de los dispositivos son móviles.

Uno de nuestros principales objetivos en el planteamiento de PDP es optimizar el coste de batería que supone la ejecución del algoritmo de descubrimiento, por lo tanto debemos reducir el número de transmisiones que realizan los dispositivos para poder descubrir servicios. En nuestro algoritmo un dispositivo sólo anuncia sus servicios si alguien se los ha solicitado, y los anuncios de los servicios se realizan por broadcast, de manera que todos los dispositivos pueden conocer la existencia de un nuevo servicio, sin necesidad de realizar una petición.

Para validar la eficiencia del algoritmo se realizarán simulaciones utilizando Network Simulator, y analizaremos como se adapta el algoritmo a diferentes escenarios de movilidad, es decir, veremos su comportamiento en caso extremos: cuando la mayoría de los dispositivos son fijos y cuando la mayoría de los dispositivos son móviles.

A continuación, describiremos formalmente el algoritmo, presentando previamente el escenario en el que lo estamos aplicando y algunas consideraciones a tener en cuenta.

A. Escenario

Nuestro entorno está formado por D dispositivos, donde cada uno de ellos tiene: un número I de interfaces de red , un número S de servicios y un tiempo de disponibilidad T. Este tiempo T ha sido configurado previamente en el dispositivo, dependiendo de sus características de movilidad. Un fragmento de DF distribuido tiene una caché asociada a cada interfaz, que contiene una lista de los servicios que se han escuchado a través de esa interfaz, cada elemento de esta lista, e, tiene dos campos: la descripción del servicio, $e.description$ y el tiempo de disponibilidad de ese servicio, $e.timeout$.

Los servicios que ofrece un dispositivo no están asociados a ningún interfaz concreto. Un dispositivo siempre anuncia sus servicios incluyendo el tiempo de disponibilidad del dispositivo, que consideramos que es una estimación del tiempo que ese dispositivo permanecerá fijo en un determinado lugar.

Cuando un dispositivo escucha un anuncio de un servicio por un interfaz lo almacena en la caché correspondiente, de tal forma que el campo $e.timeout$ tenga el valor mínimo entre el tiempo de disponibilidad con el que se anuncia el servicio, y el tiempo de disponibilidad del dispositivo local.

Por simplicidad, consideramos que los servicios locales están almacenados en la caché asociada al interfaz de loopback ($cache_0$).

B. Descripción del algoritmo

PDP tiene dos primitivas: **PDP request**, que sirve para enviar peticiones de servicios y **PDP reply**, que sirve para responder a esas peticiones, anunciando servicios disponibles.

Veremos detalladamente el funcionamiento de cada una de ellas.

B.1 PDP request

Se lanza un **PDP request** cuando las aplicaciones o el propio usuario final del dispositivo precisa un servicio, bien sea un servicio concreto, o búsqueda de servicios en general, ofrecidos por el entorno.

TABLE I
PDP request S

```
for ( i = 0  to  I  ) {
  if (S ∈ cache_i) return i;
}
for (i = 1 to I ) {
  remote_service = PDP request(S);
  if (∃ remote_service) {
    add_service(remote_service,  cache_i);
    return i;
   }
}
```

TABLE II
PDP request ALL

```
for (i = 1 to I ) {
  remote_services_list= PDP request(ALL);
}
```

Como respuesta a una petición de un servicio concreto se devolverá el interfaz por el que se puede acceder al servicio. Si la petición es de todos los servicios, realizará una actualización de todas las cachés, y la aplicación podrá consultar las diferentes cachés que posee el dispositivo para ver los servicios a los que puede acceder.

Este módulo debe minimizar el número de transmisiones por broadcast de petición de un servicio, por lo tanto:
- Si le han solicitado un servicio concreto, mirá en las cachés si existe ese servicio y en caso contrario, envía una PDP request de ese servicio.
- Si le han solicitado todos los servicios disponibles, actualiza las cachés enviando un mensaje de `PDP request` a través de todas las interfaces del dispositivo.

B.1.a PDP reply. Los fragmentos del DF distribuido en los dispositivos actúan en cada interfaz escuchando continuamente los mensajes que llegan por ella. Si estos mensajes son anuncios de servicios lo que hace es actualizar las cachés correspondientes.

Si en algún momento llega una petición de un servicio concreto S:
- Comprueba si el servicio solicitado es uno de sus servicios locales y por lo tanto, pertenece a su C_0.
- Antes de realizar `PDP reply`, genera de forma aleatoria un tiempo t, inversamente proporcional a su tiempo de disponibilidad, T; de tal forma que responderá con mayor probabilidad, aquel dispositivo que puede ofrecer el servicio durante un mayor periodo de tiempo.
- Escucha durante el tiempo t si se produce otra respuesta a la petición de servicio solicitada, si es así descarta su mensaje de respuesta, sino envía su `PDP reply`.

Si la petición es de todos los servicios, `PDP request(ALL)`:

TABLE III
PDP reply S

```
if ( ∃ e ∈ cache_0 / S = e.description) {
  t = generate_random_time(1/T);
  wait(t);
  if ( not listened PDP reply)
    PDP reply(e);
}
```

TABLE IV
PDP reply ALL

```
t = generate_random_time( 1/(T*cache_size) );
wait(t);
if ( not listened PDP reply)
  PDP reply(cache_0, cache_i);
```

- Antes de realizar `PDP reply`, genera de forma aleatoria un tiempo t, inversamente proporcional a su tiempo de disponibilidad, T, y al tamaño de su caché, $size\_C$; de tal forma que responderá con mayor probabilidad, aquel dispositivo que tenga un mayor tiempo de disponibilidad y un mayor tamaño de caché. Consideramos que este dispositivo es el que tiene una visión más acertada del mundo que le rodea.
- Escucha durante el tiempo t si se produce otra respuesta a la petición de servicios ALL, si es así, descarta su mensaje de respuesta, si no, envía un `PDP reply` con la caché de servicios locales y la caché asociada al interfaz correspondiente.

VIII. Conclusiones y trabajos futuros

En este artículo hemos descrito las bases para poder implementar un un entorno de ejecución de agentes en dispositivos limitados con J2ME, y hemos comentado cuales son los desarrollos en los que estamos trabajando en la actualidad y que inicialmente se están desarrollando en Linux. Una vez conseguida una versión estable, se realizará su migración a PalmOS y a Symbian, que nos permitirá abarcar un amplio rango de los dispositivos limitados más extendidos en el mercado: PDAs y teléfonos móviles.

Para implementar una plataforma de agentes móviles es necesario migrar los servicios básicos (páginas blancas, páginas amarillas, etc). En este artículo también hemos presentado cómo se puede implementar un servicio de páginas amarillas (Distributed Facilitator) que permita a los pequeños dispositivos en un entorno de computación ubiuca descubir servicios dinámicamente en redes ah-hoc, sin necesidad de la presencia de un servidor central. Hemos estudiado los protocolos actuales de descubrimiento de servicios y, al no ser ninguno de ellos apropiado para el caso de la computación ubicua, hemos propuesto un nuevo protocolo que denominamos PDP (Pervasive Discovery Protocol).

El siguiente paso en nuestro desarrollo, es migrar la plataforma de agentes móviles Java, TAgents, sobre nuestro perfil TAgentsP, de forma que se integren como un único proyecto en el que podemos abarcar tanto dispositivos limitados como no limitados.

En todo momento tenemos en mente desarrollar aplicaciones piloto para comprobar la viabilidad de nuestros proyectos, en concreto, como primer piloto vamos a implementar un agente "concierta citas" que es capaz de migrar a varias agendas buscando el dia y hora que le viene bien a todos los usuarios, y que automáticamente registra esa cita en todas las agendas.

References

[1] Mark Weiser. The computer for the 21st century. *Scientific American*, 9 1991.

[2] http://www.x10.com.

[3] S. Papastravrou, G. Samaras, and E. Pitoura. Mobile agents for WWW distributed database access. In *Proceedings of the International Conference on Data Engineering (ICDE99)*, 1999.

[4] M. Baldi and G.P. Picco. Evaluating the tradeoffs of mobile code design paradigms in network management applications. In R. Kemmerer, editor, *Proceedings of the 20th International Conference on Software Engineering*, IEEE CS Press, pages 146–155, April 1998.

[5] Roch H. Glitho and Samuel Pierre. Mobile agents and their use for information retrieval: A brief overview and an elaborate case study. *IEEE Network*, January 2002.

[6] Gianpaolo Cugola, Carlo Ghezzi, Gian Pietro Picco, and Giovanni Vigna. *Analyzing Mobile Code Languages. In Mobile Object Systems: Towards the Programmable Internet*, volume 1222 of *Lecture Notes in Computer Science*. Springer-Verlag, April 1997.

[7] 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.

[8] Patrik Mihailescu and Elizabeth A. Kendall. Development of an agent platform for mobile devices using J2ME. In *Proceedings of the Evolve 2001 conference*, May 2001.

[9] 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.

[10] 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.

[11] Razvan Dragomirescu. Dynamic classloading in the kvm. Technical report, Information Technology Laboratory Rochester Institute of Technology, April 2000.

[12] Alan Kaminsky. JiniME: Jini connnection technology for mobile devices. Technical report, Information Technology Laboratory Rochester Institute of Technology, August 2000.

[13] IETF Network Working Group. *Service Location Protocol*, 1997.

[14] Sun Microsystems. Jini architectural overview. white paper. Technical report, 1999.

[15] Inc. Salutation Consortium. Salutation architecture overview. Technical report, 1998.

[16] Yaron Y. Goland, Ting Cai, Paul Leach, and Ye Gu. Simple service discovery protocol/1.0. Technical report, 1999.

[17] Michael Nidd. Service Discovery in DEAPspace. *IEEE Personal Communications*, August 2001.