Agentes móviles en computación ubicua

Mª Celeste Campo Vázquez

Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid, Avda. Universidad 30
28911 Leganés, Madrid, Spain
celeste@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 estamos desarrollando actualmente se centra 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.

1 Introducción

Si nos fijamos a nuestro alrededor, la visión futurista que Mark Weiser's 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 proporcionaban 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, PDA's, 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 ofreceserlos 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 proporciona el dispositivo en el que se ejecutan, 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 de 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 distribuida que se adapta a todas estas características es lo que se denomina Agente.

Este 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 involucrando 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.

Como primer paso en nuestro análisis hemos abordado la realización de una plataforma de agentes móviles en dispositivos limitados. 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 2, 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 3, 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 estas versiones y por lo tanto, que problemas deberán solucionarse. Se describirán brevemente en esta sección también las propuestas existentes en la literatura sobre plataformas de agentes empleando J2ME. En la sección 4, describiremos detalladamente nuestra propuesta y el estado de nuestros desarrollos actuales.

2 Tecnología de agentes

La tecnología de agentes 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 sean asíncronas y que además no se precisen 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.

El concepto de agente ha sido muy discutido y aunque existen varias definiciones, la más aceptada es la que define a los agentes por sus características: movilidad, autonomía, inteligencia, comunicación, cooperación y coordinación. Un agente es aquel que posee una o varias de estas características.

2.1 Plataformas de agentes

Para introducir los conceptos relacionados con plataformas de agentes seguiremos la definición del estándar FIPA [3], que es el estándar más ampliamente reconocido y extendido en tecnología de agentes.

FIPA define una plataforma de agentes como la infraestructura, tanto hardware como software, que precisan los agentes para ser desarrollados y usados. Esta plataforma contendrá agentes especiales que proporcionarán servicios a los agentes que se encuentren en la plataforma. FIPA define tres servicios obligatorios en una plataforma de agentes:

- 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 1.

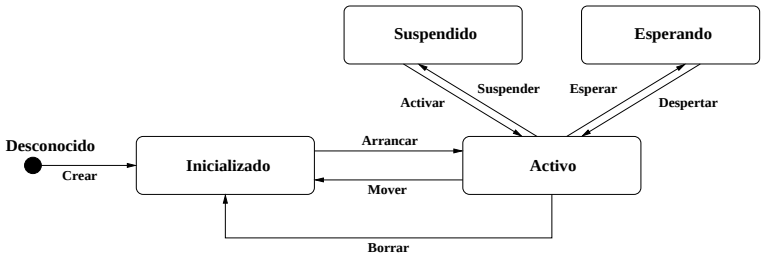

Fig. 1. Ciclo de estados de agentes

  - 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 ente agentes de una plataforma y entre agentes de distintas plataformas.

2.2 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 [4]. 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 D'Agents. 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 [5]:

- Independencia de la plataforma.
- Ejecución segura.
- Carga dinámica de clases.
- Programación multithread.
- Serialización de objetos.
- Reflexión.

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, ni proporciona control de referencias, todos los métodos públicos de un objeto Java son accesibles desde cualquier otro objeto que tenga una referencia a él.

3 Plataforma de agentes sobre dispositivos limitados

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

3.1 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.

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.

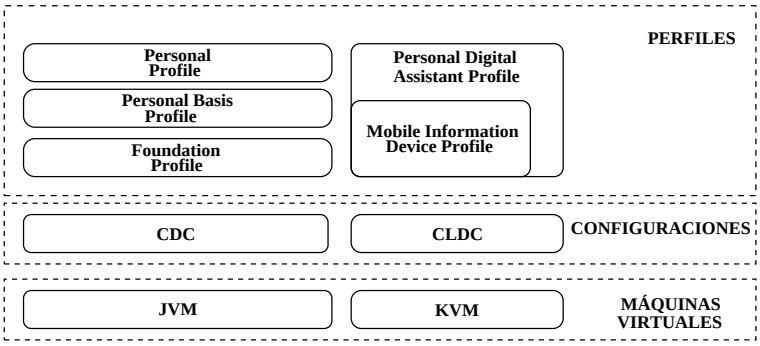

Fig. 2. Arquitectura J2ME

Una de las principales ventajas de J2ME es que es una arquitectura modular, figura 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 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.
- **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.

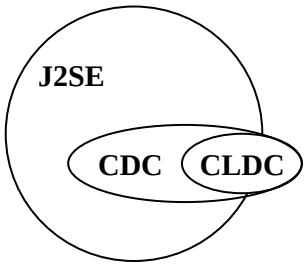

Fig. 3. Relación entre CLDC y CDC y la J2SE

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 3.

- **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.

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 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 recordamos las características que convertían a Java en un buen lenguaje para desarrollar plataformas de agentes, sección 2.2, 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 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.

LEAP El proyecto LEAP [6] 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 podrá componer la plataforma según las limitaciones del dispositivo en el que se instale.

Monash University En Monash University [7] [8] 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.

School of Computer Science of Carleton University En School of Computer Science of Carleton University [9] 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 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.

4.1 Motivación

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

Siguiendo la filosofía modular de J2ME, nuestros desarrollos actuales se centran en complementar el MIDP para construir un perfil sobre el CLDC que proporcione la funcionalidad básica de un plataforma de agentes móviles, a este perfil lo hemos llamado TAgentsP (Traveling Agents Profile), por analogía con nuestra plataforma de agentes Java, TAgents [10]. 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 (TAgent).

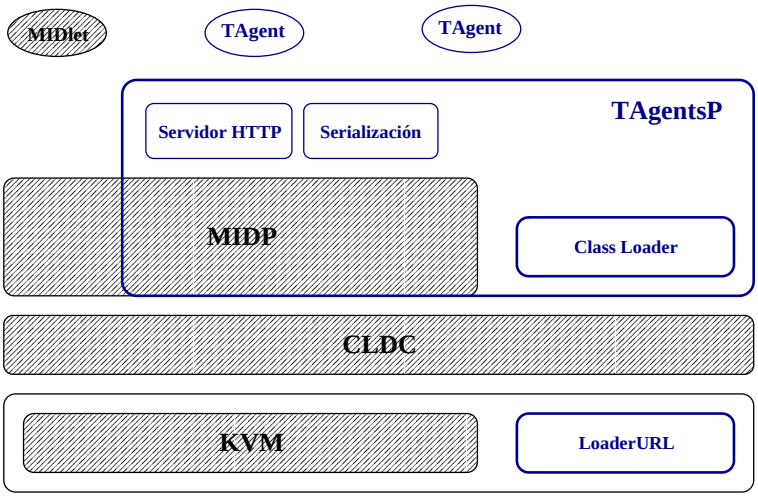

Fig. 4. Arquitectura de nuestra propuesta

En la figura 4 vemos la arquitectura de nuestra propuesta y en las siguientes secciones describiremos cada uno de los modulos que estamos desarrollando.

4.2 Carga dinámica de clases en KVM

¿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.

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.

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.

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.

Nuestra modificación de la KVM Nuestra solución añade dos modulos provisionales a la arquitectura CLDC/KVM, uno en código C, que representamos en la figura como LoaderURL, 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 URL's, siempre añadidas al final, para que no se sobreescriban clases del sistema.

El LoaderURL al ser código nativo será necesario migrarlo a las diferentes sistemas operativos que queremos involucrar en nuestro desarrollo, en concreto, se realizará su migración el PalmOS™ y en 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.

4.3 Serialización de objetos sobre CLDC

¿Para qué precisamos serialización de objetos? Solucionando el problema de la carga dinámica de clases, hemos permitido que un agente puede seguir ejecutándose en un 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.

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.

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 java.

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

4.4 Exportar ficheros de clases

¿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.

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 en la que migra el agente.

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.

5 Conclusiones

En este artículo hemos descrito las bases para poder implementar una plataforma 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: PDA's y teléfonos móviles.

El siguiente paso en nuestro desarrollo, es migrar la plataforma de agentes móviles Java, TAgents [10], 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*, September 1991.
2. 1997. http://www.x10.com.
3. FIPA. *FIPA Agent Management Specification*, March 2002.
4. 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.
5. Dany B. Lange and Mitsuru Oshima. *Programming and Developing Java Mobile Agents with Aglets*. Addison-Wesley, August 1998.
6. Lightweight extensible agent platform. http://leap.crm-paris.com/.
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 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.
9. 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.
10. Thomas Letsch. Redesign and implementation of a mobile agent system compliant with the maffinder part of masif stanadard. Master's thesis, Technische Universitat Munchen. Institut fur Informatik, 2000.
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.