Diseño de una plataforma de agentes compatible con FIPA para dispositivos limitados

Guillermo Díez-Andino Sancho, Rosa M García Rioja, Mª Celeste Campo Vázquez
Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid.
Avd. Universidad 30 28911 Leganés

e-mail:{gdandino, rgrioja, celeste}@it.uc3m.es

Abstract

Recent advances in micro-electronic and wireless technologies have fostered the proliferation of small devices with limited communication and processing power. We think that agent technology will be of great help in pervasive systems development. Pervasive systems are inherently dynamic, with devices continually coming and going. Agents are autonomous software entities that can interact with their environment, and therefore they adapt well to such frequent changes. However, the use of Multi-Agent Systems (MAS) in pervasive environments poses important challenges, and it is necessary to adapt their design to meet these challenges.

The first part of this paper describes the design of a FIPA-compliant agent platform, adapted to the limited devices that work in an ad-hoc network. The authors have presented their proposals to the Working Group FIPA Adhoc, and have been included in the white paper elaborated by this group previous to the standardization. The second part describes the agent platform implementation in real devices, using the Java 2 Micro Edition technology.

1 Introducción

En los últimos años los avances realizados en los campos de la microlectrónica y de los protocolos inalámbricos, han permitido la proliferación de un gran número de pequeños dispositivos con capacidad de cómputo y comunicación. Los más visibles son los nuevos dispositivos personales, como PDAs o teléfonos móviles, pero hay otros muchos que se embeben en el entorno que nos rodea, de forma invisible, pero que al comunicarse con nuestros dispositivos personales, permiten que el entorno físico que nos rodea se adapte a nuestras preferencias y necesidades de forma casi transparente. Esta nueva era de la computación es lo que Mark Weiser describió en su artículo "The Computer for the 21st Century" en 1991 [1] como computación ubicua.

En estos entornos existe una mayor diversidad de dispositivos, con diferentes limitaciones hardware para ejecutar aplicaciones, además gracias a los protocolos de comunicación inalámbricos se forman redes en las que los dispositivos entran y salen de forma espontánea, este tipo de redes se denominan redes ad-hoc. Tanto por las limitaciones de los dispositivos, como por las características de las redes ad-hoc, parece claro que no es eficente migrar soluciones tradicionales a estos nuevos sistemas [2]. Además, la mayoría de estos nuevos dispositivos no son multipropósito, sino que proporcionan una serie de servicios concretos, pero cuando forman una red ad-hoc entre ellos, es posible que se compongan estos servicios de forma inteligente, para ofrecer un nuevo servicio que satisfaga las expectativas del usuario de manera transparente al mismo, como quería Weiser.

En este escenario, las aplicaciones software de estos dispositivos limitados deben adaptarse a las restricciones de memoria y procesamiento que proporciona el dispositivo en el que se ejecutan, a una comunicación intermitente y de calidad cambiante, necesitan autonomía 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, o la conectividad directa no le permita alcanzar. El paradigma de computación distribuida que se adapta a todas estas características es lo que se denomina Agente [3].

Los desarrollos llevados a cabo a nivel agentes se realizaron para redes como Internet, las plataformas se ejecutaban en PCs sin limitaciones en cuanto a su capacidad de proceso y comunicación, y aunque en su definición se consideraba que se adaptaban a entornos cambiantes, con conectividad intermitente y calidad cambiante, la realidad es que estos estados eran tratados más como excepciones a las que el sistema sabía adaptarse, que como las características habituales del entorno en el que se ejecutaban.

Nuestra trabajo actual se basa en emplear el paradigma de agentes móviles como tecnología middleware para el desarrollo de servicios en redes ad-hoc formadas por dispositivos limitados,

que se comunican de forma directa sin necesidad de sistemas centrales. Para ello, tomando como punto de partida el estándar FIPA hemos realizado un nuevo diseño de plataforma de agentes móviles adaptada a los nuevos requisitos, con el mínimo impacto en la especificación. Como tecnología para apoyar la viabilidad de implantación de nuestro diseño, utilizaremos la versión de Java para dispositivos limitados, J2ME.

En la sección 2 realizaremos una breve revisión del estándar FIPA. Después en la sección 3 justificamos y proponemos simplificaciones a esta arquitectura para adaptarla tanto a las características de las redes ad-hoc, como a las restricciones de los sistemas limitados en los que se implantaría. En la sección 4 introducimos brevemente la plataforma software J2ME y analizamos la viabilidad de implementar la arquitectura diseñada sobre ella, describiendo parte del desarrollo que hemos realizado hasta la actualidad. Por último, finalizamos con las conclusiones y líneas futuras de nuestro trabajo, sección 5.

2 Estándar FIPA

Foundation for Intelligent Physical Agents[1] (FIPA) comenzó sus actividades en 1995 con el objetivo de estandizar aspectos relacionados con la tecnología de agentes y sistemas multiagente. En la actualidad se puede considerar el estándar más ampliamente reconocido y extendido internacionalmente, y se ha convertido en un referente a seguir a la hora de realizar desarrollos basados en agentes.

Las especificaciones FIPA se han agrupado en tres tipos: *component*, que se encargan de estandarizar todas las tecnologías básicas relacionadas con agentes, *informative* que describen posibles soluciones en aplicaciones realizadas con agentes en un determinado dominio y *profiles* que son conjuntos de especificaciones de tipo *component* que permiten validar cuándo una implementación es conforme al estándar.

En *FIPA Agent Management Specification* [4] se describe el modelo de referencia FIPA de plataforma de agentes y se describe la funcionalidad de cada uno de sus componentes, ver Figura 1. Estos componentes son:

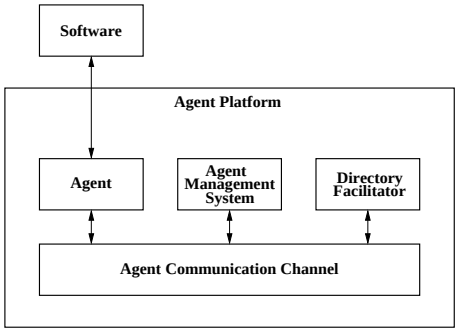

Figura 1: Arquitectura abstracta FIPA

[1] http://www.fipa.org

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

- **Directory Facilitator** (DF) : que proporciona un servicio de páginas amarillas, que permite localizar agentes por sus capacidades y no por su nombre.

- **Agent Communication Channel** (ACC): que gestiona el envío de mensajes entre agentes de la misma plataforma o de plataformas distintas, y permite la migración de agentes.

3 Diseño de la plataforma

Como hemos visto en el apartado anterior FIPA define en su arquitectura tres elementos funcionales, que proporcionan una serie de servicios básicos para los agentes que residen en la plataforma. Para que una plataforma sea compatible con FIPA debe tener obligatoriamente estos componentes, y proporcionar las interfaces y funcionalidad definida en sus estándares para cada uno de ellos.

A lo largo de este apartado analizamos cómo se puede simplificar esta funcionalidad para facilitar su implantación en dispositivos limitados, y para que el servicio que ofrecen se adapte a los requisitos impuestos por el entorno cambiante en el que se encuentran.

3.1 AMS

El Agent Management System (AMS) gestiona el ciclo de vida de los agentes, incluidos los relacionados con movilidad, es decir, crea y borra agentes y gestiona su paso de una plataforma a otra. Además el AMS da soporte a un servicio de búsqueda denominado de "páginas blancas", es decir, localizar agentes por su nombre.

Este elemento es fundamental en una plataforma de agentes, y en nuestro diseño actual lo mantenemos con la funcionalidad que se define en el estándar FIPA, pero reduciéndola a un ámbito local en la plataforma, es decir, eliminamos la posibilidad de que el AMS extienda sus búsquedas de agentes a otras plataformas, poniéndose en contacto con AMS remotos.

3.2 DF

La adaptación de la funcionalidad del DF para dispositivos limitados que operan en redes ad-hoc es uno de los temas más importantes de investigación que se está llevando a cabo. El Working Group

AdHoc de FIPA centra su trabajo actual, exclusivamente, en la adaptación del DF. De hecho la propuesta que se describe en este apartado es una de las que están siendo consideradas en el grupo de trabajo [5].

El DF es el elemento de la plataforma que le permite a un agente descubrir los servicios que ofrecen otros agentes en el entorno que le rodea. En la definición del funcionamiento de un DF tradicional, la búsqueda de servicios remotos se realiza utilizando el concepto de federación de DFs. Un DF además de tener registrados unos servicios proporcionados por agentes locales (nativos o no), puede tener registrados otros DF, lo que le permite poder extender sus búsquedas a los servicios que tienen registrados esos otros DF. El número de saltos entre DF federados se limita mediante un parámetro de restricciones de búsqueda.

Analizamos a continuación los inconvenientes de la federación de DF para el descubrimiento de servicios remotos en redes ad-hoc:

- La búsqueda de un servicio remoto concreto siempre implica una comunicación con este sistema, porque a priori sólo conocemos que en esa plataforma existe un DF y no los servicios que están registrados en él. Lo que implica realizar transmisiones sin garantía de éxito.

- Si las condiciones de búsqueda permiten saltos de más de un nivel, es decir, la búsqueda en un DF remoto puede extenderse a otros DF remotos federados en él. Puede ocurrir, al contrario que en redes tradicionales, que los servicios que están registrados en esos DF no están accesibles al sistema inicial y por lo tanto, no sean resultados válidos.

3.3 Ad-hoc Discovery Agent

Para solventar estos dos problemas expuestos, y con el objetivo de mantener la mismas funciones que se especifican en FIPA00023 para el DF, de forma que los agentes no necesiten modificar la forma en la que interaccionan con él para descubrir servicios. Proponemos la introducción en la arquitectura FIPA para redes ad-hoc de un nuevo agente el **Ad-hoc Discovery Agent (ADA)** obligatorio en la plataforma, con la siguiente funcionalidad:

- Registrará y mantendrá actualizados, en el DF de la plataforma en la que se encuentre, los servicios remotos ofrecidos en la red ad-hoc. Es decir, el DF tendrá entradas a servicios remotos y no a un DF remoto, de esta manera, minimizaremos el número de transmisiones [2].

- Propagará búsquedas de servicios remotos solicitadas al DF, cuando éste se lo solicite, porque las condiciones de búsqueda así lo permiten.

- Anunciará a través del protocolo de descubrimiento asociado, lo servicios registrado en el DF, cuando esté permitido.

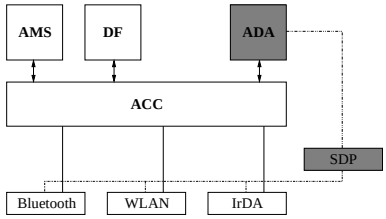

Figura 2: Introducción de ADA en FIPA

ADA utilizará un Service Discovery Protocol (SDP) para descubrir los servicios remotos. En este sentido deberá seleccionarse un protocolo que se adapte a la restricciones expuestas en el apartado anterior, para garantizar una solución eficiente en redes ad-hoc. En [6] se realiza un análisis de los posibles protocolos que se pueden emplear y se propone la utilización de Pervasive Discovery Protocol (PDP), que ha sido definido en nuestro grupo de trabajo. El Working Group FIPA AdHoc ha publicado un White Paper en el que se incluye parte de nuestro análisis y la descripción de nuestro protocolo [7].

3.3.1 Registro del ADA en el DF

El agente ADA proporciona un servicio que obligatoriamente tiene que estar registrado en el DF de la plataforma en la que reside, de esta forma en un DF que opere en una red ad-hoc siempre existirá una entrada correspondiente al ADA, que poseerá el siguiente AID reservado:

```
(agent-identifier
  :name ada@hap
  :addresses (sequence hap_transport_address)
)
```

Este agente se registra en el DF poniendo en el parámetro `:type` del `service-description` el valor `ada`.

El DF cuando procesa un `search` y en las restricciones de la búsqueda se le permite propagar la búsqueda a la red ad-hoc (se puede reutilizar la restricción `max-depth = 1`) el DF delega la búsqueda al ADA.

El ADA cada vez que tenga conocimiento a través del SDP de algún nuevo servicio remoto en la red ad-hoc, lo dará de alta en el DF mediante una petición de `register`. Como la red es cambiante estos registros tendrán un `timeout` asociado que gestionará el ADA de tal forma que cuando este expire realizará una petición de `deregister` al DF. El ADA sólo debe almacenar el `agent-identifier` correspondiente y su `timeout` asociado.

[2] siempre y cuando el protocolo de descubrimiento de servicios que emplee este agente se adapte a entornos ad-hoc y realice el mínimo número de transmisiones para descubrir servicios remotos

El ADA a su vez puede proporcionar al SDP si éste lo solicita los servicios locales registrados en el DF, para ello realizará una petición de `search` sobre el DF.

3.4 ACC

En una sociedad multiagente, los agentes deben cooperar para realizar sus tareas y alcanzar sus objetivos. Los mecanismos de comunicación tradicionales permiten que dos agentes puedan transferirse información, pero es insuficiente cuando el objetivo es el de conseguir un comportamiento social. Es decir, se requieren mecanismos que permitan dotar a los mensajes intercambiados de un contenido semántico.

FIPA ha desarrollado el lenguaje ACL (Agent Comunication Language), una evolución de KQML, tomado como estándar durante mucho tiempo para el intercambio de conocimiento entre los agentes. La definición semántica formal con lógica modal de los mensajes ACL ha sido uno de los esfuerzos mayores dentro del estándar FIPA y otorga a este lenguaje de una gran aceptación como estándar dentro del mundo de los agentes. El elemento ACC de la arquitectura FIPA es el que da soporte a la comunicación entre agentes empleando ACL. Para su simplificación hemos realizado un análisis de ACL, que presentamos en este apartado.

ACL esta basado en **actos comunicativos** que se encargan del paso de información, solicitud de la información, negociación, realización de acciones y manejo de errores (Se explican de forma más detallada en el siguiente subapartado).

Las conversaciones entre agentes en ocasiones siguen patrones determinados, que se repiten en muchos casos. Aprovechando estos patrones y su repetición, FIPA ha definido unos **protocolos**. Un protocolo es un patrón que se usa para llevar por unos cauces concretos una conversación. Son como conversaciones guiadas, en que cada agente sabe qué mensaje enviar y cuáles puede recibir.

3.4.1 Comunicacion entre agentes

La idea central de los actos comunicativos entre los agentes gira entorno a la solicitud de la realización de una acción a otro agente. Esta operación se puede realizar de diferentes formas dependiendo si la acción se solicita a un agente concreto o si no se sabe quién provee esa acción.

Cuando un agente A solicita a otro agente concreto B una acción, envía un mensaje `request`. El destinatario puede aceptar o rechazar la petición con un mensaje `accept` o `refuse` respectivamente. En caso de aceptarla, deberá realizar la acción, e indicárselo al agente A cuando finalice mediante un mensaje de tipo `agree`. Si se produce un fallo en la realización de la acción se le notifica al agente que inició la petición mediante el mensaje `failure`.

Si el agente A necesita que se realice una operación pero no conoce quién tiene esa habilidad o se sabe que hay varios agentes que la proveen, se establece una comunicación de negociación. En este caso, se inicia la comunicación con el mensaje tipo `cfp`, pidiendo propuestas a los distintos agentes. En la petición se especifica la acción a realizar y algunas precondiciones a la hora de hacer las propuestas. Los agentes consultados envían sus propuestas al agente mediante `propose`, también indicando las condiciones de la propuesta y si no la aceptan lo notifican con el envío de un mensaje `refuse`. Para evitar que el agente A espere mucho tiempo se establece un tiempo límite para la respuestas descartando las que lleguen después de éste. El agente A estudia las propuestas y elige las que le convienen, notificándoselo a los agentes mediante `accept-proposal` y rechazando el resto con `reject-proposal`. Una vez aceptadas y rechazadas las propuestas, el agente que inició el proceso puede cancelarlo si se produce algún cambio sobre la situación inicial. Así mismo, una vez aceptada una propuesta el agente B puede responder con un `inform` indicando que se ha realizado la acción, o indicando que ha habido algún fallo con un `failure`.

Los agentes también se comunican para intercambiar información. De modo que el agente inicia la solicitud de información con `query-if` o `query-ref`, tras la petición de información, el agente B puede responder con la propia información (`inform`), con un fallo (`failure`), con un `not-understood` o con el rechazo de la petición, teniendo que alegar el motivo.

Todo lo comentado anteriormente hace referencia a la comunicación entre los agentes, pero ACL también se emplea para solicitar operaciones a los elementos integrantes de la plataforma, como por ejemplo el AMS y el DF.

3.4.2 Estructura del mensaje

El mensaje ACL FIPA contiene uno o más elementos de mensaje necesarios para la comunicación eficiente entre los agentes. Se compone de un **identificador** del tipo del acto comunicativo que define el significado del mensaje o la acción que el agente emisor solicita con dicho mensaje. También incluye una secuencia de **parámetros** definidos como un conjunto de parejas clave-valor que permiten asociar a cada acto de comunicación concreto toda la información necesaria.

accept-proposal	agree	cancel
cfp(call for proposal)	confirm	disconfirm
failure	inform	inform-it
inform-ref	not-understood	propagate
propose	proxy	query-if
query-ref	request	request-when
request-whenever	subscribe	

Figura 3: Mensajes ACL

3.4.3 Conjunto minimal de mensajes ACL

Como se ha visto en el apartado anterior, ACL cuenta con gran variedad de mensajes, además cada uno de ellos tiene hasta 11 parámetros diferentes, lo que hace que sea un lenguaje pesado para ser implementado completamente en una plataforma de agentes móviles para dispositivos limitados. Este requisito ha hecho que se seleccione un conjunto de mensajes mínimo que provea las necesidades de comunicación entre los agentes.

La necesidad de petición de operaciones se puede cubrir con los mensajes `request`, `inform` y `agree`. De igual manera que las peticiones y la solicitud de información a los elementos que componen la plataforma.

El caso de la negociación explicado con anterioridad requiere muchos mensajes, pero se puede simplificar de la siguiente manera. Al agente A envía un `request` al AMS o al DF solicitando los agentes que realizan la operación en concreto. Los elementos de la plataforma envían la información mediante un mensaje `inform`, y de esta forma se ha obtenido la funcionlidad de `cfp` y de `proposal`. El siguiente paso es decidir a quién se le requiere el desempeño de esa operación, esta tarea se realiza de forma interna en el propio agente. Una vez que se ha decidido a quién realizar la petición se envía el `request` y se procede igual que en las peticiones de operación.

En lo referente a la solicitud de información también se puede realizar mediante peticiones `request` y luego mediante los parámetros especificar si se requiere la ejecución de una operación o si simplemente se solicita una información en concreto permitiendo así sustituir el mensaje `query` por `request`.

Con lo que se concluye que el grupo mínimo de mensajes para comunicarse en una plataforma en dispositivos limitados es `request`, `inform`, `agree`. El mensaje `cancel` se puede sustituir por el establecimiento de un tiempo de respuesta.

3.4.4 Ejemplo de mensaje ACL

A continuación se presenta un mensaje que envía un agente al AMS para registrarse en esa plataforma. En el mensaje se incluye el identificador del receptor mediante el parámetro `receiver` y adjunta la información relativa a sí mismo con el parámetro `sender`. Después se indica el lenguaje, el protocolo y la ontología, para continuar con el contenido del mensaje donde se especifica la operación que se está solicitando, en este caso consiste en registrarse y se indica mediante `register`.

```
(request
  :sender
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name ams@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (action
      (agent-identifier
        :name ams@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (register
        (ams-agent-description
          :name
            (agent-identifier
              :name dummy@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :state active))))
```

4 Implementación

En paralelo al diseño de una plataforma compatible con FIPA, nuestro grupo de trabajo comenzó a evaluar la posibilidad de poder implementar esta plataforma en dispositivos reales.

Este estudio nos llevo en primer lugar, a la elección de un lenguaje de programación para realizar la implementación. A día de hoy podemos decir que aunque se han desarrollado lenguajes específicos, ha sido Java el lenguaje en el que más plataformas de agentes móviles se han desarrollado, debido a las siguientes ventajas: independencia de la plataforma, ejecución segura, carga dinámica de clases, serialización, reflexión,...

Esto nos llevó a seleccionar la versión de Java para dispositivos limitados J2ME, y en concreto MIDP, como base de nuestros desarrollos. Aunque J2ME mantiene alguna de las características de Java, parte de las que convertían a Java en un buen lenguaje de programación de plataformas de agentes móviles, 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). Veremos a lo largo de esta sección cómo hemos solventado estos problemas.

4.1 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.

4.1.1 LEAP

El proyecto LEAP [8] 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 que pueda operar tanto en dispositivos limitados (teléfonos móviles, PDA, pagers) con J2ME, como en PCs 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 puede componer la plataforma según las limitaciones del dispositivo en el que se instale.

El proyecto LEAP fue pionero a la hora de demostrar la viabilidad de construir plataformas de agentes en dispositivos limitados, aunque su aplicabilidad está centrada en redes con infraestructura y por lo tanto, en los terminales móviles no existía una plataforma completa de agentes como tal y su operación dependía siempre de un sistema intermedio al que estuviese conectado.

Aunque era uno de los objetivos marcados en el proyecto, la movilidad a nivel agente no está soportada.

4.1.2 Monash University

En Monash University [9] se ha diseñado e implementado una plataforma de agentes para dispositivos personales móviles basados en PalmOS.

La plataforma desarrollada se denomina MAE y en la actualidad tienen dos implementaciones utilizando dos versiones reducidas de Java: la configuración CLDC de J2ME, y SuperWaba, que es una versión limitada de Java con algunas funcionalidades no soportadas por la versión oficial de Sun, como JNI.

Esta plataforma soporta movilidad a nivel agente, para solventar el problema de la carga dinámica de clases se han empleando mecanismos específicos de los sistemas PalmOS, por lo que esta característica no es migrable a otro tipo de dispositivos J2ME.

4.2 TAgentsP: nuestro perfil sobre CLDC para agentes móviles

Las propuestas anteriores han realizado importantes contribuciones en cuanto a la adaptación de plataformas de agentes en dispositivos limitados. LEAP es una plataforma válida para una red con infraestructura, en la que no es necesario tener un plataforma autónoma en el dispositivo limitado, y por lo tanto se puede depender de un sistema no limitado, pero el escenario que nosotros queremos abordar, que es una red ad-hoc esto no es válido. MAE es una plataforma completa pero no es compatible con FIPA y funciona sólo sobre PalmOS.

Nuestro investigación y desarrollo inicial se centró en 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*: serialización de objetos y carga dinámica de clases.

Para ello, siguiendo la filosofía modular de J2ME [10] complementamos el perfil MIDP para construir un nuevo perfil sobre el CLDC que proporcione la funcionalidad básica de un plataforma de agentes móviles, a este perfil lo hemos llamado TAgentsP (Travel Agents Profile), por analogía con nuestra plataforma de agentes Java, TAgents [11].

Una vez proporcionados los servicios básicos, y para conseguir nuestro objetivo de implementar una plataforma de agentes móviles autónoma, es decir sin necesidad de interaccionar con dispositivos no limitados, hemos desarrollado un reducido servidor HTTP que nos permite por una parte la movilidad de agentes y por otra, la comunicación entre ellos de forma directa.

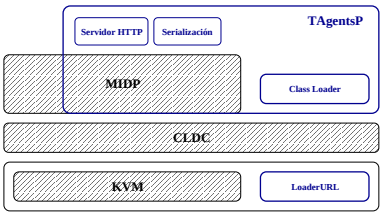

Figura 4: Arquitectura de TAgentsP

En la Figura 4 vemos la arquitectura de nuestra propuesta y en las siguientes secciones describiremos cada uno de los módulos desarrollados y probados con éxito en dispositivos limitados: el módulo de serialización y el del servidor HTTP. Todavía no hemos abordado la implementación de la carga dinámica aunque expondremos brevemente el porqué.

4.2.1 Implementación de TAgentsP

Una de las primeras carencias que se pretenden suplir en nuestro nuevo perfil TAgentsP es la ausencia de mecanismos de carga dinámica de clases y de serialización de objetos, que nos permitan la movilidad de código entre dispositivos limitados.

La movilidad de un agente se consigue transformando un agente en un flujo de bytes que viaja a través de la red, para posteriormente ser reconstruido en el dispositivo destino, este mecanismo se conoce como serialización. Pero además, a la hora de soportar la migración de agentes es necesario proporcionar carga dinámica de clases, ya que el sistema destino si no posee la clase a la que pertenece el agente serializado recibido, éste no podría ser reconstruido.

En J2ME no se dispone de un mecanismo de carga dinámica de clases, aunque es una de las funcionalidades que se pretenden suplir en próximas versiones. La implementación de mecanismos de carga dinámica de clases, implica modificar la implementación nativa de la máquina virtual, por lo que este mecanismo no es portable a distintos sistemas. Éste es el principal motivo que nos ha llevado a postergar esta implementación y suplirla introduciendo el mismo conjunto de clases en los sistemas limitados entre los que realizamos movilidad de código.

En las próximas secciones proporcionamos una descripción del mecanismo de serialización implementado y del servidor HTTP contruido.

Mecanismo de serialización en J2ME La *serialización* es un mecanismo mediante el que se puede convertir un objeto en un flujo de bytes que represente su estado, y consecuentemente ser transportado a través de la red o almacenado de manera persistente en un sistema de ficheros. Esta conversión no tendría sentido si no fuese a haber

una posterior recuperación, mecanismo denominado *deserialización*.

La serialización es un requisito imprescindible para soportar la movilidad de agentes, ya que es el mecanismo que permite convertir a un agente en algo transportable que conserve su estado. Además, también es un mecanismo necesario para la comunicación de mensajes entre agentes residentes en distintas plataformas. Java en su versión estándar proporciona serialización, pero en J2ME es una de las características que se han eliminado, aunque si bien plantea serias dificultades de implementación es abordable y además ofrece un mecanismo abierto y extensible en cuanto a la movilidad de código se refiere.

Debido a la ausencia del mecanismo de reflexión en J2ME, por el cual las propias clases son capaces de inspeccionarse a sí mismas (averiguando sus métodos, atributos, parámetros, constructores), el mecanismo de serialización implementado va a tener ciertas limitaciones, no pudiéndose conseguir una serialización totalmente automatizada, en la que el programador no tenga que intervenir en este proceso.

Se puede decir que lo que se busca conseguir, es el soporte básico para que mediante éste y el conocimiento del programador sobre las clases que implementa se disponga de un mecanismo genérico y extensible para transformar objetos en un flujo de bytes, transportarlos y posteriormente recuperar su estado en el dispositivo destino.

El principal problema a afrontar a la hora de desarrollar mecanismos de serialización es la imposibilidad de conocer en tiempo de ejecución los métodos y atributos de las clases en J2ME, que se proporcionaba en las versiones estándar de Java.

La serialización llevada a cabo facilitará y se encargará de que el programador se despreocupe de cómo se lleva a cabo todo el proceso, del formato interno de los datos así como del proceso de recuperación de las clases serializadas, siendo únicamente responsable de la implementación del interfaz `Serializable` como se verá a continuación.

La solución planteada consiste en primer lugar en la creación de un interfaz denominado `Serializable` que contenga los dos métodos básicos que especifiquen las operaciones que toda clase serializable debe poder llevar a cabo: `writeObject` para serializarse y `readObject` para deserializarse. Éste interfaz tiene el objetivo de poder "marcar" aquéllas clases que sí saben cómo serializarse/deserializarse.

El siguiente paso es la creación de dos nuevas clases `ObjInputStream` y `ObjOutputStream`. Estas clases son las que van a controlar el proceso completo de serialización/deserialización realizando las llamadas necesarias a los métodos `writeObject` y `readObject` que todas las clases serializables deben implementar.

El método `writeObject` se va a encargar de escribir los valores de los atributos de un determinado objeto en un flujo de bytes, obteniendo de este modo un objeto serializado, que mediante el método `readObject` podrá ser interpretado permitiendo la construcción de un nuevo objeto a partir de la información extraida de este objeto serializado.

En el momento de serializar una determinada clase, por ejemplo `Agente_Serializable` se va a crear un objeto de tipo `ObjOutputStream`. Este objeto recibirá el objeto a serializar, creará un flujo de bytes en el que inicialmente escribirá el nombre de la clase del objeto que se está serializando[3] para finalmente llamar al método `writeObject` de la clase `Agente_Serializable` que se preocupará de escribir la información precisa (valor de atributos, tipos, ...) en un flujo de bytes de modo que posteriormente pueda recuperar su estado empleando para ello su propio método `readObject`.

Este mecanismo ofrece la ventaja de que aún careciendo de mecanismos de reflexión si el programador se preocupa de implementar los métodos de serialización y deserialización (writeObject y readObject) toda clase en J2ME que implemente el interfaz serializable puede ser convertida en un único flujo de bytes que va a poder viajar de dispositivo en dispositivo, en donde con una simple llamada al método readObject este flujo se convierta en un nuevo objeto con el mismo estado del objeto serializado de manera totalmente transparente.

Servidor HTTP en dispositivos limitados

El perfil TAgentsP desarrollado no tendría utilidad alguna si no existiese un mecanismo de comunicación directa entre los distintos dispositivos móviles. En nuestro desarrollo se ha optado por construir un reducido servidor HTTP 1.1 en J2ME como mecanismo de comunicación directa entre dispositivos ya que ofrece una solución abierta no orientada únicamente al intercambio de clases o agentes serializados.

A continuación se introduce la arquitectura básica del servidor. Esta arquitectura está compuesto por tres módulos principalmente. El primero de ellos es el de *configuración*; mediante éste se definen las opciones básicas de funcionamiento (número de peticiones a aceptar, puerto, nombre del servidor, archivos MIME soportados ...).

A través del *sistema de archivos* (segundo módulo) se van a poder gestionar los recursos a albergar por el servidor así como resolver las peticiones de los usuarios. Por último es mediante el *servicio de tratamiento de peticiones* que las diversas solicitudes HTTP (GET,POST,HEAD) van a poder ser tratadas.

Uno de los problemas surgidos en el desarrollo de este servidor ha sido la ausencia de un sistema de archivos accesible en J2ME para poder proporcionar los recursos solicitados por los clientes HTTP. La solución ha consistido en implementar

[3] llamando a Object.getClass().getName() soportado en J2ME

un pseudo sistema de archivos sobre RMS [4] de modo que puedan crearse directorios y archivos así como gestionarlos (recuperar, eliminar, listar ...).

Este servidor podrá utilizarse para diversas tareas como pueden ser: intercambio de clases entre diferentes dispositivos que permite el soporte a la movilidad de agentes y la comunicación directa entre dispositivos sobre HTTP, por ejemplo como mecanismo de transporte de mensajes ACL.

5 Conclusiones y trabajos futuros

En este artículo proponemos la utilización de la tecnología de agentes móviles como middleware para el desarrollo de aplicaciones en dispositivos limitados que se comunican mediante protocolos inalámbricos, formando lo que se denomina red ad-hoc. Siendo ésta una contribución a la computación ubicua, permitiendo integrar a los dispositivos limitados y embebidos en el mundo físico, en la computación distribuida.

Tomando como punto de partida el estándar FIPA, hemos realizado un análisis de los diferentes elementos funcionales que se definen y hemos propuesto una simplificación para que puedan implementarse en un dispositivo limitado. En el caso del DF, no sólo es necesario su simplificación sino su adaptación a las restricciones de las redes ad-hoc. En este sentido, los autores forman parte activa del Working Group FIPA AdHoc, en el que en la actualidad se están discutiendo las propuestas relacionadas con el DF, entre ellas, la aquí propuesta.

En la actualidad, seguimos trabajando en el diseño de la plataforma definiendo la interfaz entre el agente ADA y el protocolo de descubrimiento de servicios suyacente y el propio DF. Y seguimos analizando el impacto que la simplificación de ACL puede tener en la comunicación entre agentes. También dentro de nuestro grupo estamos definiendo un modelo de seguridad aplicable a sistemas multiagente en redes ad-hoc.

Como hemos visto, para implementar el diseño realizado hemos seleccionado la tecnología J2ME y hemos definido e implementado un nuevo perfil J2ME, TAgentsP, que nos dé el soporte para movilidad de código y comunicación directa entre dispositivos que J2ME no proporcionaba. Este perfil ha sido probado en dispositivos reales con resultados satisfactorios. Nuestro trabajo actual en esta línea se centra en implementar la plataforma de agentes, integrando nuestros desarrollos en la plataforma LEAP.

Agradecimientos

Los autores agradecen a David Camacho, Carlos García Rubio y Andrés Marín las aportaciones realizadas.

Celeste Campo da las gracias también a Florina Almenares y a los miembros de FIPA TC/WG Adhoc, especialmente a Michael Berger.

References

[1] Mark Weiser. The Computer for the 21st Century. *Scientific American*, September 1991.

[2] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Deborra Zukowshi. Challenges: An Application Model for Pervasive Computing. In *Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, 2000.

[3] H. S. Nwana. Software Agents: an overview. *Knowledge Engineering Review*, 1(3):205–244, 1996.

[4] FIPA. *FIPA Agent Management Specification*, March 2002.

[5] C. Campo. Directory Facilitator and Service Discovery Agent. Technical report, FIPA Adhoc Technical Committee, 2002.

[6] C. Campo. Service Discovery in Pervasive Multi-Agent Systems. In Tim Finin and Zakaria Maamar, editors, *AAMAS Workshop on Ubiquitous Agents on embedded, wearable, and mobile agents*, Bologna, Italy, July 2002.

[7] WG-AdHoc. Agents in Ad Hoc Environments. A Whitepaper, 2002.

[8] Lightweight extensible agent platform. http://leap.crm-paris.com/.

[9] Patrik Mihailescu and Elizabeth A. Kendall. MAE: A Mobile Agent Platform for Building Wireless M-Commerce Applications. In *8th ECOOP Workshop on Mobile Object Systems: Agent Applications and New Frontiers*, Spain, June 2002.

[10] Enrique C. Ortiz. A Survey of J2ME Today. Technical report, Wireless Java, 2002.

[11] 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.

[4] Paquete de almacenamiento persistente en J2ME que permite la creación y almacenamiento de registros binarios de datos.