HERMES - PLATAFORMA DE DESARROLLO DE AGENTES MOVILES PARA DISPOSITIVOS MOVILES

Proyecto de Trabajo de Grado

José Luís Jurado Muñoz
Cristian Eduardo Martínez Collazos

Director: Dr. Juan Carlos Vidal
UNIVERSIDAD DEL CAUCA

UNIVERSIDAD DEL CAUCA
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas

Popayán, 2007

TABLA DE CONTENIDO

LISTA DE FIGURAS

RESUMEN

El proyecto HERMES es una herramienta software con fines académicos, que ofrece al desarrollador de aplicaciones una API para la implementación de agentes móviles en dispositivos móviles y fijos. HERMES es un prototipo de plataforma de agentes móviles que permite crear ambientes de ejecución para agentes móviles de tal modo que se desarrolle su ciclo de vida.

La necesidad de ofrecer nuevos servicios en las aplicaciones residentes en dispositivos móviles, pretenden aumentar la funcionalidad de las mismas implementando mecanismos que busquen el procesamiento remoto, baja demanda en el ancho de banda, migración de procesos, entre otros. Pero existe un incoveniente para los desarrolladores que deseen implemtar este tipo de aplicaciones y es que aun persiste la insuficiencia en el soporte para el desarrollo de aplicaciones de este tipo. De este modo la plataforma HERMES busca brindar una herramienta para la implementación de aplicaciones móviles que puedan desarrollar funcionalidades acordes a los problemas plateados anteriormente.

El uso de la tecnología de agentes móviles implementado en las aplicaciones residentes en dispositivos móviles, es el aporte de HERMES a problemas como los mencionados anteriormente. Ademas gracias a la capcidad de migración que ofrecen los agentes móviles, una aplicación movil que antes se encontraba en desventaja con una aplicación fija en cuanto a la falta de recursos computacionales (Hardware y Software) podra realizar operaciones de alta demanda computacional gracias al uso de la distribución de sus tareas. El soporte para realizar este tipo de aplicaciones lo ofrece HERMES.

Algunos proyectos no tan conocidos como TAgentsP[1], MAE[2] y AURA[3] que fueron desarrollados con fines académicos al igual que HERMES, por grupos de desarrollo de distintas universidades permiten la movilidad de sus agentes en dispositivos móviles, pero los mecanismos utilizados para implementar dicha movilidad obligan a tener una versión exclusiva de la maquina virtual de la plataforma de soporte JME, ya que la implementación que se ha hecho es una extensión a la maquina virtual de JME de JAVA. Por lo tanto resulta inconveniente

[1] TAgentsP: Referirse al capitulo II (Estado del Arte).
[2] MAE: Referirse al capitulo II (Estado del Arte).
[3] AURA: Referirse al capitulo II (Estado del Arte).

la utilización de estos proyectos por que carecen de portabilidad y suficiente soporte para el estudio de los mismos.

El beneficio que ofrece HERMES es que tiene una API propia con un lenguaje sencillo y funcional de sus componentes, permite implementar agentes móviles en dispositivos móviles y fijos. Ofrece la capacidad de manejar las interrupciones del proceso de migración por inconvenientes en la conexión, para enviar y recibir agentes móviles entre dispositivos móviles y fijos. Ademas brinda la posibilidad de crear canales de comunicación remotos y locales entre sus agentes.

El proyecto HERMES se desarrolló en tres ciclos, el primero de ellos fue el proceso de investigación acerca de la viabilidad de implantar agentes móviles en dispositivos móviles, luego se procedió a enriquecer los conocimientos acerca de agentes móviles y su aporte a la tecnología de dispositivos móviles. El siguiente paso durante este proceso de investigación fue recopilar toda la información necesaria sobre proyectos similares o afines a los propósitos que se querían cumplir con el fin de encontrar falencias y desventajas de lo ya existente en el campo de agentes móviles y procurar mejorar y aportar nuevas y mejores ideas a estudios e implementaciones existentes.

El segundo ciclo se concentró en diseñar una arquitectura funcional y escalable que permitiera a nuevos proyectos mejorar y optimizar lo desarrollado por HERMES. El desarrollo y pruebas de los componentes que poco a poco se implementaban buscaban comprobar cada servicio y componente desarrollado. De este modo cada componente comprobado era necesario para nuevos servicios que se fueran implementando en la plataforma. Al final de este ciclo se obtuvo una arquitectura en tres niveles (entorno, ASH y Sistema de Soporte)[4] que diera el soporte básico a la implementación de ambiente de ejecución de agentes móviles, donde se permitiera la migración entre dispositivos móviles a fijos y viceversa.

El último ciclo desarrollado a lo largo del proyecto HERMES, se centró en la implementación de diferentes pruebas que permitieran conocer el manejo y funcionalidad de la API de HERMES. Cada prueba realizada buscaba comprobar un servicio específico de HERMES y una última prueba comprobó la correcta utilización y funcionalidad de HERMES en casos reales. Los resultados de cada prueba hecha fueron satisfactorios y comprobaron la funcionalidad y desempeño de HERMES.

[4] Remitirse al capitulo III (Descripción de la Plataforma HERMES).

INTRODUCCION

El desarrollo de aplicaciones, servicios y soluciones para dispositivos móviles es un área de trabajo que ofrece un amplio rango de opciones a los desarrolladores y clientes tanto en hardware como en software. La movilidad es quizás la característica más relevante y atractiva que estos dispositivos inalámbricos ofrecen al usuario.

Las aplicaciones móviles buscan maximizar y potencializar las características de los dispositivos móviles, pero se encuentran en desventaja con las aplicaciones residentes en equipos fijos por la insuficiencia en sus recursos (Hardware y software). Resulta entonces inconveniente y poco práctico desarrollar aplicaciones para un dispositivo móvil que demande una serie de servicios y recursos con los cuales no se cuenta, como alto rendimiento en el procesamiento de tareas, disponibilidad de recursos locales y remotos, etc. Servicios que muchos diseñadores quisieran implementar en sus aplicaciones móviles. Desafortunadamente no existe una plataforma específica que de soporte a las necesidades anteriormente mencionadas para el desarrollo de aplicaciones en dispositivos móviles.

Se busca que una plataforma ofrezca soporte a ciertas necesidades que exigen los desarrolladores de aplicaciones, como mejorar la capacidad de procesamiento de tareas, capacidad de migración de los procesos, delegación de tareas, baja demanda de ancho de banda, y otras más, que un dispositivo móvil requeriría para mejorar el desempeño de sus aplicaciones.

La tecnología de agentes surge entonces como la idea más práctica y óptima para alcanzar el propósito de mejorar el desempeño y funcionalidad de las aplicaciones desarrolladas en los dispositivos móviles. La justificación y motivación de implantar agentes móviles en las aplicaciones para dispositivos móviles, se basa en la facilidad de la aplicación de agentes en la computación distribuida. La ganancia 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 no precisen conexiones permanentes para la ejecución de tareas [Bigus 2002].

Otras de las razones para utilizar agentes móviles en las aplicaciones para dispositivos móviles, se basa en que los agentes móviles poseen características

que le otorgan capacidades básicas relacionadas con la autonomía del agente, la flexibilidad de su comportamiento, movilidad e interacción con otros agentes. Quizás una de las características más importantes de los agentes es el nivel de autonomía que poseen, tal propiedad es una característica inherente que reduce significativamente el acoplamiento entre componentes de un servicio. Además las comunicaciones se realizan a través de mensajes sin necesidad de un contacto estrecho entre ellos, es decir cada agente puede seguir ejecutando sus funciones mientras realiza un proceso de comunicación. Estas y otras propiedades son las que se buscan explorar y explotar en el desarrollo de servicios de aplicaciones en dispositivos móviles basados en la tecnología de agentes móviles [Franklin 1996].

Una aplicación que se desarrolle con agentes móviles se convierte en una herramienta útil para el desarrollo de servicios distribuidos, aportando nuevas capacidades que aun no han sido exploradas y que pueden contribuir valiosamente a estudios a nivel teórico y tecnológico, trazando un camino para la evolución natural que estos sistemas deben experimentar. [Ferguson 1992]

Existen varios proyectos de vanguardia en torno al concepto de agentes móviles como Aglets[5], Jade[6], ARA[7], entre otros. Pero el inconveniente que presentan estos proyectos es que no existe soporte para migrar agentes entre dispositivos móviles y fijos. Este hecho motiva el desarrollo de un ambiente de implementación que permita maximizar y mejorar los servicios de las aplicaciones en los dispositivos móviles y fijos. De este modo el presente trabajo presenta a HERMES (Plataforma de Desarrollo de Sistema MultiAgente para Dispositivos Móviles), la cual brindará un ambiente para desarrollar aplicaciones con Agentes Móviles en Dispositivos Móviles y fijos.

Con el desarrollo de HERMES se busca aumentar las posibilidades y funcionalidad del diseño de servicios basados en el paradigma de agentes móviles, de modo que considere a los dispositivos móviles como los terminales que intervienen y acceden a la funcionalidad provista por un sistema de Agentes Móviles.

Gracias a la posibilidad de migrar agentes de un dispositivo móvil a un dispositivo fijo y viceversa, las Aplicaciones MultiAgente que se desarrollen con la plataforma HERMES permiten al diseñador de aplicaciones aumentar la gama de servicios ofrecidos. Maximizando los recursos y beneficios que un dispositivo móvil y un fijo puedan llegar a prestar a los usuarios finales.

[5] Agltes: Referirse a la sección 1.4.1 del capítulo I (Marco Teórico).
[6] Jade: Referirse a la sección 1.4.4 del Capítulo I (Marco Teórico).
[7] ARA: Referirse a la sección 1.4.5 del Capítulo I (Marco Teórico).

Por estas razones, este proyecto de grado ofrece una alternativa de solución a los problemas mencionados a través de la ejecución de los siguientes objetivos.

Objetivo General

- Diseñar e implementar una plataforma software denominada HERMES, para el desarrollo de aplicaciones con Agentes Móviles para dispositivos móviles.

Objetivos Específicos

- Diseñar y desarrollar los paquetes de Interfaces de Programación de Aplicaciones necesarios para soportar el desarrollo de Aplicaciones con agentes móviles.

- Diseñar y desarrollar el ambiente de ejecución y servicios de soporte para la migración de agentes móviles en dispositivos de cómputo (fijos y/o móviles).

- Diseñar e implementar aplicaciones prototipo para validar la plataforma Hermes

A lo largo del presente documento se abordan los conceptos teóricos y prácticos que se necesitaron para llevar a cabo el proyecto. El documento se encuentra organizado como se explica a continuación.

Capítulo I Marco Teórico

Este capítulo contiene las bases teóricas sobre las cuales se encuentra enmarcado el proyecto: Desarrollo de aplicaciones en Dispositivos móviles, Plataformas de Servicios JME y para JSE, Antecedentes Teóricos, Plataformas de Agentes Móviles, Tecnologías de Agentes Móviles, Conceptos Básicos y Desarrollo de Sistemas MultiAgentes.

Capítulo II Estado del Arte

Este capítulo contiene un resumen acerca de algunos de los proyectos realizados por universidades y particulares en torno al concepto de agentes móviles.

Capítulo III Descripción del Sistema HERMES

En este capítulo se detalla cada uno de los componentes y servicios integrados en la Plataforma HERMES, además del uso y adaptación del paradigma de agentes móviles y sistemas multiAgentes.

Capítulo IV Ambiente Experimental

Este capítulo describe en qué consiste y cómo se desarrolla cada una de las pruebas hechas para confrontar la correcta funcionalidad y desempeño de cada uno de los componentes desarrollados en el sistema HERMES. Así como los resultados obtenidos en la validación de cada componente probado.

Capítulo V Conclusiones y Trabajo a Futuro

Este capítulo describe las conclusiones a las cuales se haya llegado después de desarrollar el trabajo por completo. También contiene recomendaciones para futuros trabajos sobre nuevas versiones en la plataforma HERMES.

Bibliografía y Glosario

Este capítulo presenta las referencias a los documentos utilizados y el respectivo glosario.

Anexos

Este capítulo contiene la documentación que profundiza y precisa el contenido del documento del proyecto final.

CAPITULO I

MARCO TEORICO

1.1. Desarrollo para Aplicaciones en Dispositivos Móviles

Las tecnologías de la información y de las comunicaciones están experimentando una evolución exponencial, tanto desde el punto de vista de la oferta de productos y servicios como desde la demanda. Este hecho es particularmente significativo en el ámbito de las comunicaciones móviles y sistemas inalámbricos, que progresivamente se están integrando en el mundo Internet, dando lugar a una auténtica revolución. Las tecnologías inalámbricas están, por tanto, jugando un papel fundamental en la nueva Sociedad de la Información y del Conocimiento. Posibilitando la comunicación instantánea desde cualquier lugar, fijo o móvil.

A medida que vaya incrementándose el uso de los servicios “persona a máquina”, “maquina a persona”, relacionados con el acceso a información, se estimulará aún más esta conectividad entre personas y por lo tanto las aplicaciones que se desarrollen para cumplir con estos servicios serán en mayor demanda y tendrán que suplir las necesidades en muchos campos y espacios que se utilicen estos dispositivos. [Habitat 2005]

El mercado ha comenzado realmente a tomar en cuenta los servicios móviles como un nuevo canal para la evolución del negocio. Por otro lado, la automatización de procesos se ha convertido en una necesidad reiterada de aquellas empresas que buscan reducir costes y agilizar el negocio.

Las principales aplicaciones desde el punto de vista de la movilidad, que se están abordando en determinados sectores son las siguientes [Telefonica 2005]:

- *El sector de las comunicaciones interpersonales.*
- *El sector del ocio y entretenimiento.*

- *El sector de la seguridad.*
- *El sector del transporte.*
- *El sector de los seguros.*
- *El sector del comercio electrónico.*
- *Las Administraciones Públicas.*

La movilidad enfocada al campo empresarial permite dotar a los trabajadores de una mayor libertad, más allá de la mensajería electrónica. La fuerza de ventas y el concepto del tele trabajo pueden disponer de funcionalidad completa aunque no estén en su oficina, lo que aumentará la efectividad en las ventas de la empresa y permitirá nuevas formas de trabajo. [Telefonica 2005]

De igual modo existen un sin número de servicios orientados hacia la movilidad que son aplicables a sectores relacionados con:

- Los servicios domóticos.
- Los servicios M2M (Machine to Machine).
- La fuerza de ventas.
- El teletrabajo.

El sector del ocio electrónico es uno de los mercados que con más rapidez adoptó el uso de las tecnologías móviles para ampliar su campo de explotación. El auge del uso lúdico de la tecnología SMS animó a proveedores, fabricantes y operadores móviles a ampliar la oferta de ocio electrónico. Durante los últimos años han proliferado los servicios de descargas de tonos y logos, chats por SMS/WAP y juegos móviles sobre múltiples plataformas tecnológicas, como la iniciativa JME de Sun MicroSystems. [Donald 2002]

La posibilidad de ofrecer diversos servicios en cualquier lugar (sin tener que desplazarse a las oficinas de la administración) facilita las labores para los ciudadanos. Así mismo, los servicios basados en la movilidad permiten la agilización de algunos procedimientos que pueden ser más costosos si se realizan de forma tradicional.

La disponibilidad de este tipo de servicios permite que los empleados, por medio de una simple aplicación, puedan hacer un exacto seguimiento del proceso de ventas. Las operadoras, mediante sus servicios de conectividad, localización y mensajería, pueden proporcionar a la fuerza de ventas la posibilidad de realizar consultas online de las rutas diarias, o de la información relativa a productos (stocks, tarifas, etc.) o clientes (estado de riesgo, pedidos pendientes, etc.), agilizando el rendimiento del equipo comercial de cualquier organización, ya que dispone de todos estos servicios y datos sin necesidad de volver a la oficina.

Entre los avances que conlleva la movilidad asociada al puesto de trabajo, hay que destacar el ahorro de tiempo, dinero y energía. Otro aspecto positivo es que la movilidad puede ayudar a incorporar al mercado laboral a personas con discapacidad física. Además, aumenta nuevas oportunidades de trabajo. [Douglas 2001]

Se puede observar entonces la gran cantidad de opciones que tienen las aplicaciones móviles al ofrecer un sin número de servicios en distintos campos de la vida diaria y las opciones crecerán a medida que las necesidades y cambios culturales, sociales, económicos y religiosos así lo determinen. Convirtiéndose el desarrollo de aplicaciones móviles en un campo aun muy poco aprovechado pero con mucho futuro.

1.2. Plataformas y Sistemas Operativos para Dispositivos Móviles

Las plataformas y sistemas opertativos descritos a continuación fueron utilizadas como base para la implementacion de la plataforma HERMES en el caso de JME la importancia de su uso es descrita en capitulos posteriores. Sistemas operativos como symbian, Palm OS y Windows Mobile, Linux para móviles es necesario conocer un poco sobre su funcionalidad y descripción en general por que seran los sistemas operativos donde se instale la plataforma HERMES y donde las aplicaciones desarrolladas con HERMES puedan ser ejecutadas.

1.2.1. Plataforma JME

La compañía Sun Microsystem[8] lanzó a mediados de los años 90 la tecnología Java, la cual, gracias al paradigma “Write Once, Run Anywhere”, se ha convertido en una de las tecnologías dominante para el desarrollo de aplicaciones empresariales, según datos suministrados por estudios de consultaría de la empresa Sun Microsystem. Ahora bien, JME (Java Micro Edition) es la realización de la plataforma Java orientada a dispositivos con prestaciones inferiores a las de un PC, como es el caso de los teléfonos móviles que soportan JME. Forma parte de la versión 2 de Java, que incluye JEE (Java Enterprise Edition) para servidores y JSE (Java Standard Edition) para ordenadores de sobremesa. La principal ventaja de JME es su independencia de la plataforma, lo que ha supuesto que prácticamente todos los fabricantes de terminales móviles, independientemente del SO empleado en el terminal, incorporen JME para la ejecución de aplicaciones

[8] Sun Microsystem: **Sun Microsystems** es una empresa informática del Silicon Valley, fabricante de semiconductores y software, referirse a [Wiki 2006].

desarrolladas por terceros. Actualmente, excepto en los terminales de gama baja, la inmensa mayoría de los fabricantes de terminales incluyen esta tecnología. [SUN 2005]

Arquitectura

JME se estructura en configuraciones y perfiles, como lo muestra la figura 1. Una configuración contiene la máquina virtual Java (JVM) y un conjunto de clases genéricas ("configuración") que pueden ser empleadas en un amplio rango de dispositivos. Por su parte, el "perfil" contiene las clases que permiten el desarrollo de las aplicaciones en un tipo de dispositivo determinado. Hay especificadas varias configuraciones y perfiles. Sin embargo, en los terminales móviles se emplea la combinación de la configuración CLDC (Connected Limited Device Configuration) y el perfil MIDP (Mobile Information Device Profile).

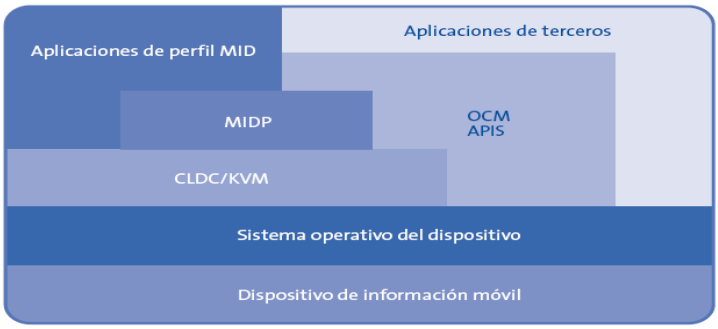

Ilustración 1 Arquitectura de JME

La Configuración CLDC

La configuración CLDC sólo contiene la máquina virtual JVM y las clases Java más básicas. Por tanto, esta configuración no se encarga de aspectos como la gestión del ciclo de vida de la aplicación, la gestión de la interfaz de usuario o la gestión de los eventos que se llevan a cabo con las clases del perfil. La mayor parte de las clases que componen la configuración CLDC son heredadas de J2SE, tal es el caso de las funciones básicas de E/S, los tipos de datos, el manejo de errores y excepciones y las utilidades básicas. [Mun 2000]

También dispone de una serie de clases específicas que, de manera generalizada y expansible, definen funcionalidades de E/S y comunicaciones. El conjunto de estas clases forma lo que se conoce como Generic Connection Framework (GCF). Con objeto de establecer la base para todas las comunicaciones en JME, la

configuración CLDC introduce el concepto de GCF. En este caso, la implementación del protocolo es específica de cada tipo de plataforma, por eso GCF no implementa nada, sólo define de forma abstracta y genérica cómo manejar los distintos tipos de conexiones. La implementación de los protocolos debe ser realizada por el perfil MIDP o por las clases específicas añadidas por el fabricante en sus dispositivos. La clase principal de GCF se utiliza para realizar los distintos tipos de conexiones relacionados con:

- Los archivos.
- El dispositivo de entrada y salida (puerto serial).
- Las comunicaciones en modo datagrama.
- Las comunicaciones TCP.
- La conexión básica a un servidor Web, a través de HTTP

Perfiles

En la arquitectura de JME, por encima de la configuración, tenemos el perfil (*profile*). El perfil es un grupo más específico de APIs, desde el punto de vista del dispositivo. Es decir, la configuración se ajusta a una familia de dispositivos, y el perfil se orienta hacia un grupo determinado de dispositivos dentro de dicha familia. El perfil, añade funcionalidades adicionales a las proporcionadas por la configuración. [SUN 2005]

Perfil MIDP 1.0

La primera versión del perfil MIDP define un conjunto de APIs específicas cuyo objetivo es facilitar el desarrollo de las aplicaciones JME. Sus características principales son:

La interfaz de usuario
El concepto central de la interfaz de usuario en MIDP 1.0 es el de “pantalla” (screen), que es un objeto que encapsula una serie de gráficos específicos del dispositivo representados a través de la interfaz de usuario. Desde un punto de vista lógico, la interfaz de usuario está compuesta de dos APIs, la API de alto nivel y de bajo nivel.

El ciclo de vida de la aplicación
MIDP 1.0 define un modelo de aplicación para permitir que los recursos del sistema sean compartidos por varios MIDlets (aplicaciones MIDP) cómo debe ser empaquetado, qué entorno de ejecución tiene disponible y cómo debe

comportarse. Así como también es responsable de la gestión de los errores durante la instalación, así como de la ejecución o eliminación de la aplicación y de las interacciones necesarias con el usuario.

El mecanismo de persistencia
MIDP 1.0 ofrece un mecanismo denominado RMS (Record Management System) para que los MIDlets puedan almacenar y recuperar datos. Los registros son guardados en un almacén de registros representado por la clase RecordStore.

Las funciones de conexión
MIDP soporta mediante este paquete un subconjunto de elementos del protocolo HTTP, que puede ser implementado con otros protocolos IP (como TCP/IP) o con otros protocolos que no son IP (como WAP o i- Mode), utilizando un gateway para permitir el acceso a los servidores HTTP. También define los métodos necesarios para establecer las conexiones HTTP y realizar peticiones a los servidores Web.

El mecanismo de seguridad
La seguridad en JME está centrada en dos áreas:

- *La seguridad de bajo nivel (maquina virtual)*: Garantizar que la aplicación no sea capaz de dañar el dispositivo en el que se está ejecutando.

- *La seguridad de nivel de aplicación*: se basa en restringir las clases que puede emplear una aplicación respecto a las definidas en la configuración, en los perfiles y en las propias definiciones del dispositivo, de modo que no se pueden emplear las clases definidas por el usuario.

Perfil MIDP 2.0

Las APIs provistas en MIDP 1.0 eran bastante limitadas, y, aparte de las características básicas del lenguaje Java, prácticamente no ofrecen más que una arquitectura rudimentaria de interfaz gráfica y la posibilidad de acceder a información remota mediante HTTP. Por lo tanto se adicionó nuevas funcionalidades a esta versión, entre sus principales características están:

- Compatibilidad hacia atrás con MIDP 1.0.

- Dispone de una interfaz avanzada de usuario, que tiene mayor control de la pantalla y portabilidad, y también incluye nuevos elementos gráficos (pop-up, scroll, text-box, formularios u objetos definidos por el usuario) mejorados.

- Dispone de soporte multimedia: tonos, secuencias de tonos, ficheros WAV y soporte para videostreaming.
- Dispone de soporte avanzado para juegos, aprovechando mejor la capacidad nativa de los terminales.
- Dispone de conectividad avanzada, incluyendo HTTPS y soporte push para la activación de MIDlets.
- Dispone de provisión OTA (Over-The-Air), para la descarga de MIDlets.
- Dispone de seguridad extremo a extremo: HTTPS, SSL y WTLS. [SUN 2005]

1.2.2. Sistema Operativo Symbian

Actualmente Symbian es un consorcio propiedad de algunos de los principales fabricantes de terminales móviles a escala mundial: Nokia[9], Panasonic[5], Samsung[5], Siemens[5] y Sony-Ericsson[5]. El uso del sistema operativo (SO Symbian) realizado por este consorcio no está limitado exclusivamente a los propietarios, sino que cualquier otra compañía puede adquirir una licencia para incorporar dicho SO a sus terminales.

Características Funcionales de las Versiones Actuales

Las versiones más novedosas de Symbian están por la línea de su versión 7.0 y 8.0 aunque muchos terminales que actualmente están en el mercado siguen la línea de la versión 6.0 y sus derivados. En la figura 2 se ilustra la arquitectura del sistema operativo Symbian, y la dependencia entre los componentes del sistema (los componentes de los niveles más altos dependen de los componentes situados en los niveles inferiores). [Castañeda 2004]

[9] Nokia, Panasonic, Samsung, Siemens, Sony-Ericson : Referirse a [Telefonica 2005]

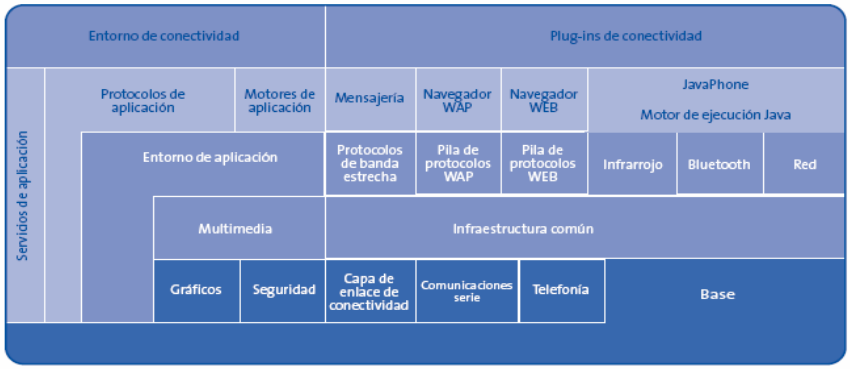

Ilustración 2. ***Arquitectura del Sistema Operativo Symbian.***

Los componentes y características más importantes a destacar de las versiones actuales y sus derivados son:

- *La base*. Incluye los componentes básicos de todo sistema operativo: kernel, gestión de la memoria y de los procesos, sistema de archivos, manejadores de dispositivos, seguridad de bajo nivel, librerías básicas de usuario, etc.

- *El gestor de telefonía*. Realiza la Gestión de los sistemas móviles celulares.

- *Las pilas de infraestructura de comunicaciones y red*. Incluyen la infraestructura de comunicaciones TCP/IP, GSM, GPRS, WAP, infrarrojos, Bluetooth y comunicaciones serie.

- *El gestor multimedia*. Se encarga de gestionar tanto la reproducción y grabación del audio como las diferentes funcionalidades de la imagen.

- *El gestor de la seguridad*. Se encarga de gestionar la seguridad relacionada con la descarga e instalación de las aplicaciones.

- *La aplicación "framework*". Contiene las librerías de middleware para la gestión de los datos, el texto, los portapapeles, los gráficos, la internacionalización y los componentes básicos de la interfaz gráfica de usuario.

- *El gestor de mensajería*. Realiza la gestión del correo electrónico, de los mensajes de texto y del fax.

- *Los motores de la aplicación*. Realizan la gestión de los contactos, la agenda, las tareas y otras aplicaciones.

- *El entorno de ejecución Java*.

- *El correo electrónico* (POP3, IMAP4, SMTP, MHTML) dispone de soporte para archivos adjuntos (incluyendo documentos de Word), fax y mensajería SMS.

- Protocolos de telefonía de 2G (voz y datos por circuitos) y 2,5G (datos por paquetes).

- *Protocolos de comunicación*: TCP/IP, WAP, Bluetooth, IrDA, Serial.

- *Encriptación y gestión de los certificados*. Se dispone de protocolos seguros de comunicación (HTTPS, WTLS y SSL), con objeto de realizar la instalación de las aplicaciones basadas en certificados.

- *Motores de aplicación* (contactos, agenda, mensajería, navegación, voz, etc.).

- Intercambio de objetos mediante el protocolo OBEX.

- Se dispone de soporte para varios formatos de audio y vídeo. Así como soporte para codificación Unicode.

- Se dispone de varias opciones para programación y desarrollo de contenido: C++, Java, WAP y Web.

- Es posible realizar la sincronización de los datos con el PC, utilizando Symbian Connect.

- Existe soporte para varias interfaces de usuario (teléfonos móviles basados en teclado y pantalla táctil con lápiz, y teléfonos móviles con capacidades avanzadas de datos).

- Motores de navegación Web completa.

- Soporte para MMS y EMS.

- Librerías de aceleración de gráficos, streaming y acceso directo a la pantalla.

- Dispone de IPv6 (con IPSec) y USB.

- Librería abstracta para estándares de comunicaciones móviles 2G, 2,5G y 3G.

- *Sincronización OTA* (Over The Air) basada en el protocolo SyncML, con objeto de realizar la sincronización de los datos de información personal (PIM) y la conversión de documentos.

- Soporte para J2ME MIDP 1.0. y MIDP 2.0.

- Un entorno multimedia con soporte para múltiples threads.

- El protocolo de comunicaciones Bluetooth 1.1.

- Soporte a las últimas tecnologías y versiones de los estándares, tales como J2ME (CLCD1.1/MIDP2.0), IPv6 y DM-OTA. También soporta las últimas arquitecturas de CPU y periféricos, e incorpora un kernel multithread en tiempo real que mejorará las prestaciones multimedia de los terminales. [Mun 2000]

1.2.3. Sistema Operativo Windows Mobile

Todos los sistemas operativos o plataformas de Microsoft para dispositivos con capacidades más reducidas que los PCs derivan de Windows CE, que a su vez proviene de la especificación Win32 empleada en PC, y se denominan genéricamente Windows Mobile.

La plataforma Windows Mobile for Pocket PC es un subconjunto bastante amplio de la API de Windows CE, y está orientada a dispositivos tipo PDA dotados de microprocesadores potentes y grandes pantallas.

Hay otra plataforma, conocida como Windows Mobile for Smartphone, que también es una adaptación de Windows CE, pero que, sin embargo, es más restrictiva, ya que está adaptada a teléfonos móviles. Smartphone especifica con alto grado de detalle las capacidades suministradas por el dispositivo, el modo de interacción con el usuario y las dimensiones de la pantalla gráfica (176 x 220 píxeles en la configuración preferente). Por esta razón, una aplicación diseñada

para un dispositivo Smartphone concreto funcionará de forma correcta en otro dispositivo que disponga de este sistema operativo, sin necesidad de realizar cambios (de forma similar a lo que ocurre con las plataformas Symbian Series 60, 70, etc.). [Perez 2000]

Además de las funciones propias de un SO heredadas de Windows CE, Smartphone proporciona una serie de APIs orientadas al desarrollo de aplicaciones en teléfonos móviles, como son:

- La gestión y sincronización de agendas y listas de contactos.
- El establecimiento de conexiones de voz y datos.
- La conectividad HTTP.
- La navegación WAP.

Por otro lado, la construcción de las interfaces gráficas se apoya en controles de interfaz similares a los de Windows (cajas de edición, listas, botones, controles estáticos, etc.). Uno de los mayores atractivos que tiene esta plataforma para el usuario final es que, junto al sistema operativo, se incluyen de fábrica diferentes versiones adaptadas de algunos de los programas de ofimática más populares de Microsoft, como: Internet Explorer y MSN Messenger. El entorno de programación usado para el desarrollo de aplicaciones sobre Smartphone es Embedded Visual C++, que es funcionalmente muy similar a Visual Studio y que se utiliza de forma generalizada para la programación de aplicaciones en sistemas Windows.

La promoción de este sistema operativo por parte de Microsoft se ha encontrado con obstáculos técnicos y comerciales (a nadie se le oculta que Microsoft pretende reproducir en los teléfonos el modelo de negocio de Windows en los PCs). Además, los grandes fabricantes de teléfonos móviles, como Nokia, Samsung o Sony-Ericsson, han rechazado de forma generalizada aliarse con Microsoft en esta iniciativa. Por el contrario, la plataforma de Pocket PC se ha convertido en el sistema operativo líder en los dispositivos PDA y WDA, partiendo de una situación de desventaja frente a las PDAs de Palm.

El entorno Windows CE.NET

Windows CE.NET no es un lenguaje de programación sino un entorno de ejecución sobre los sistemas operativos de la familia Windows CE, con características muy similares a la tecnología Java. Sin embargo, uno de los aspectos más interesantes de esta tecnología es que la naturaleza interpretada

del entorno de ejecución, soportada por la CLR (Common Language Run-time), permite el desarrollo de aplicaciones en una amplia variedad de lenguajes de programación, como Managed C++ (versión especial de C++ para .NET), VB.NET y, sobre todo, C#. En este sentido, Windows CE.NET busca aprovechar sinergias con el entorno .NET de Microsoft para los sistemas operativos de la familia Windows, de forma que se posibilite la reutilización de código y que se reduzca la barrera de entrada de los nuevos programadores. Como ocurre con Smartphone, Windows CE.NET está siendo usado por un número muy limitado de fabricantes en dispositivos que no están destinados al gran público, fundamentalmente por las razones de estrategia comercial mencionadas anteriormente. [MICR 2000]

1.2.4. Sistema Operativo Palm OS

PalmOS es el sistema operativo que desarrolló el fabricante de PDAs Palm, actualmente PalmOne; y en estos momentos ha sido transferido a una empresa independiente denominada PalmSource. Aunque hay algunos fabricantes, como es el caso de Handspring (actualmente integrado en PalmOne), que usan PalmOS en los equipos que disponen de capacidad de comunicación móvil incorporada, mayoritariamente este SO se emplea en aquellas agendas que no integran conexión a una red móvil. No obstante, las agendas pueden tener capacidad de conexión empleando un teléfono auxiliar que se conecta a ellas a través de una interfaz IrDA o Bluetooth.

PalmOS es un sistema operativo completo, como puede ser Symbian, que además de las funciones habituales de un SO proporciona acceso a todas las capacidades del dispositivo (desde la gestión de información del usuario hasta la sincronización y el control de las conexiones de datos por redes celulares). Además, PalmOS define un modelo extremadamente sencillo de ejecución concurrente de aplicaciones, basado en una emulación poco costosa de técnicas de multithreading. El sistema de archivos de PalmOS no es equiparable al de un computador personal, sino que se estructura en una base de datos simple de registros asociados a las distintas aplicaciones. Estas soluciones son radicalmente distintas a las adoptadas en los sistemas operativos de computadores personales, lo que complica, hasta cierto punto, el aprendizaje de los nuevos programadores. Las aplicaciones PalmOS tradicionalmente tienen una ocupación en memoria muy reducida en comparación con su competidor Microsoft Pocket PC, en línea con la relativamente baja capacidad computacional y de almacenamiento de esta plataforma. Con PalmOS, las capacidades gráficas y funcionales de los dispositivos Palm se equiparan a las PDAs de gama alta, basadas en soluciones de Microsoft. Con todo ello, la cuota de mercado de los dispositivos Palm, que en

años pasados era mayoritaria, sobre todo en Estados Unidos, ha ido decreciendo en favor de Pocket PC.

Este SO es un sistema abierto, provisto de avanzados entornos de desarrollo integrados (incluyendo depuración paso a paso) y SDKs para el desarrollo de aplicaciones por terceros. La programación se realiza en leguaje C++.

El desarrollo del sistema se ha dividido en dos versiones diferentes:

- *La versión de Cobalt:* Para PDAs de altas prestaciones. Esta versión dispone de unos requisitos de memoria de 256 Mbyte de RAM y 256 Mbyte de ROM, y está basado en la tecnología de BeOS (un sistema operativo para PCs desarrollado por antiguos empleados de Apple).

- *La versión de Garnet:* Para teléfonos inteligentes. Esta versión dispone de unos requisitos de memoria de 128 Mbyte de RAM y 16 Mbyte de ROM. [Telefónica 2005]

1.2.5. Sistema Operativo Linux para Dispositivos Móviles

Existe un gran mercado en expansión para los dispositivos empotrados, que incluye consolas de información y dispositivos móviles. Hay varias opciones de software para sistemas empotrados, que incluyen DOS, Microsoft Windows y Linux. Actualmente Linux es el sistema operativo mejor preparado para los sistemas empotrados de tiempo real. El sistema operativo Linux ofrece muchas ventajas para los servidores empotrados, pues se puede portar a muchas de las CPUs y plataformas hardware existentes, es estable y escalable para un amplio rango de capacidades, y su utilización por los desarrolladores es sencilla.

En los sistemas empotrados los márgenes comerciales son muy bajos, y el hecho de que Linux esté desarrollado sobre software libre ayuda a este mercado. El uso de código abierto para los sistemas empotrados evita las cuotas de licencia de los fabricantes de código cerrado, lo que implica importantes ahorros de coste en la producción de un volumen elevado de estos sistemas. En este sentido, los desarrolladores de sistemas empotrados pueden parametrizar Linux para que se ajuste a las necesidades de sus aplicaciones específicas, y permite a todos los desarrolladores poder mejorar de forma cooperativa el software y arreglar los errores en tiempo real. Linux no fue originalmente diseñado para los sistemas empotrados, pero ha sido adoptado por ellos. Además, no fue diseñado desde el principio de forma integrada, sino dividido en componentes. Por esta razón, los desarrolladores de sistemas empotrados tienen la seguridad de que sus sistemas

se pueden modificar y reparar de forma inmediata, dado que el código de Linux está siendo constantemente mejorado

Hay un riesgo de que se produzca fragmentación en el mercado de Linux para los sistemas empotrados, ya que existen más de cien sistemas operativos comerciales de tiempo real y una gran variedad de dispositivos empotrados (desde teléfonos celulares a frigoríficos). En este sentido, para adaptar Linux a estos dispositivos y poder utilizar la menor cantidad de hardware posible, se extraen diferentes partes del código básico común de GNU/Linux y se añaden diferentes extensiones específicas a los dispositivos que optimicen el rendimiento.

ELC es un consorcio de Linux para sistemas empotrados, que se constituyó como una asociación sin ánimo de lucro independiente de los fabricantes, y que se dedica a promover y hacer que Linux avance como el sistema operativo de la comunidad de los sistemas empotrados. Actualmente existe una comunidad de desarrollo de software libre que está trabajando en portar Linux a los terminales de HTC Wallaby e Himalaya (que corresponden a los modelos de Telefónica MoviStar TSM 400 y TSM 500, respectivamente). Esta comunidad lleva trabajando un año en este entorno. HTC no colabora directamente en el proyecto como lo está haciendo Compaq en su proyecto homólogo, sin embargo se han conseguido hasta la fecha avances significativos que confirman que la migración a Linux es factible. No obstante, aún no se ha completado el desarrollo de la versión Linux adaptada a los dispositivos mencionados. [Telefónica 2005]

1.3. Conceptos Básicos de Agentes

1.3.1. Definición de Agentes

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 alcanzar, reacciona ante eventos externos además puede comunicarse y colaborar con otros agentes.

Un agente es un sistema computacional capaz de actuar independientemente del usuario. En otras palabras, un agente comprende qué acciones debe ejecutar para satisfacer los objetivos para los cuales fue diseñado sin necesidad de que se le especifique lo que debe hacer en un determinado momento. [Lingau 1999]

Una definición universalmente aceptada de lo que es un agente es algo difícil de encontrar. Existe un consenso general en la comunidad informática sobre la autonomía que éstos poseen, pero contrariedad cuando se tienen en cuenta otros atributos, ya que la importancia que éstos tienen depende del dominio sobre el que operan, de esta manera, la habilidad de aprender de sus propias experiencias puede jugar un papel importante en un dominio específico, mientras que para otro puede ser completamente indeseable.

1.3.2. Agentes Móviles

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. [Brooks 1996]

Un agente móvil es un programa con características especiales que presenta tres estados; un estado de datos (estructura de datos y otros objetos no agentes), un estado de código (las clases del agente y otras referencias a objetos) y un estado de ejecución (el control de procesos que se ejecutan en el agente).

Para aclarar la definición de agente móvil es necesario precisar el concepto de 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. Migrar un agente consiste en enviar su código y el estado de ejecución a un nodo remoto, de forma que después de su migración se retome su ejecución en el mismo punto en el que se encontraba antes de migrar. [Bigus 2002]

El agente móvil accede a un conjunto de métodos que le permiten moverse de un servidor a otro, siendo ésta su característica principal. El agente posee un programa de ejecución principal que define las tareas que realizará y el grado de inteligencia con el que actuará para interactuar con los usuarios y para resolver tareas adversas en su entorno. Finalmente es autónomo porque no necesita consultar a ningún supervisor humano para tomar una decisión y porque posee un hilo de ejecución propio e independiente del resto de los agentes. [IBM 1996]

Además de la movilidad, un agente móvil puede tener las siguientes características:

- *Autonomía:* Tienen su propio hilo y actúan por sí mismos.
- *Concurrente:* Puede ejecutarse con más agentes a la vez.
- *Direccionable*: Se puede definir su comportamiento.
- *Continuo:* Se ejecutan continuamente y por tiempo indefinido.
- *Reactivo:* Reacciona a su entorno mediante métodos.
- *Social:* Interoperan con objetos y otros agentes.

1.3.3. Lugares

La tecnología de agentes modela una red de ordenadores, tan grande como una colección de lugares que ofrecen un servicio a los agentes móviles que entran en él. Así pues un lugar es un sitio de interacción de agentes que les ofrece un servicio y les facilita la consecución de sus tareas a todos aquellos agentes a los que les haya sido permitido entrar.

1.3.4. Viajes

Un viaje es la acción de recorrer diferentes nodos de una red por parte del agente móvil, se establece un itinerario de viaje para tener registro de los lugares escogidos durante el recorrido del agente por la red. A los agentes les está permitido viajar de un lugar a otro, sin importar la distancia, esto es el sello distintivo de un sistema de programación remota. De esta forma un viaje le permite a un agente utilizar un servicio ofrecido remotamente y regresar a su lugar de origen.

1.3.5. Comunicación entre Agentes

Este proceso de los agentes se asemeja a realizar reuniones (meetings) entre agentes. A dos agentes les está permitido comunicarse si ellos se encuentran en el mismo entorno. Una reunión les permite a los agentes en el mismo computador comunicarse entre ellos a través de mensajes, por tal razón se dice que hay comunicación entre agentes. Las reuniones motivan a los agentes a viajar, un agente podría viajar a un lugar en un servidor para comunicarse con un agente estacionario que provee el servicio que el lugar ofrece.

1.3.6. Conexiones

Una conexión de comunicación entre dos agentes en lugares distintos, es con frecuencia hecha en beneficio de los usuarios humanos para que interactúen con

las aplicaciones. La instrucción de conexión permite a los agentes de los usuarios el intercambio de la información a distancia. Por ejemplo un agente podría enviar información al sitio de donde proviene (a través de otro agente) para que el usuario origen pueda seleccionar alguna opción de la información que ha encontrado.

1.3.7. Autorizaciones

La autoridad de un agente o lugar en el mundo electrónico es el individuo, organización entidad o empresa a quien representa dentro del mundo físico. Una autoridad puede ser propietaria de varios agentes, la autenticación de los agentes generalmente consiste en descubrir su autoridad. Los agentes y los lugares pueden discernir, pero nunca podrán negar ni falsificar sus autoridades, evitando el anonimato.

1.3.8. Permisos

Las autoridades pueden limitar que los agentes y lugares puedan hacer asignaciones y permisos a ellos mismos. Un permiso es un dato que acepta capacidades. Un agente o lugar puede darse cuenta de sus capacidades, lo que le es permitido hacer, pero nunca podrá incrementarlas. [Celeste 2004]

1.4. Plataformas de Agentes Móviles

Desde el principio de la tecnología de agentes móviles se han creado diversas plataformas para su desarrollo. Las primeras quizá carecieron de muchos requerimientos que exigen las aplicaciones reales, tal vez fue la razón por la que no tuvieron éxito, sin embargo sentaron las bases y los conceptos de esta tecnología que cada día ha ido mejorando.

Existe una gran cantidad de plataformas para el desarrollo agentes móviles, todas, si bien siguen la misma filosofía, difieren en la creación de agentes así como en los protocolos de transmisión.

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

- Protocolos de Conexión: Que permita ocultar el manejo e implementación del protocolo de comunicación que habilita a dos máquinas para que se comuniquen. Las máquinas se comunican para transportar agentes. Los protocolos deben operar sobre una gran cantidad de redes de transporte incluyendo aquellas basadas en el protocolo de Internet TCP/IP. [Ferguson 1992]

- Servidores de Agentes: Un agente requiere un servidor de aplicaciones, un servidor de agentes, que debe estar ejecutándose en un dispositivo móvil o fijo, antes de que el agente pueda visitarlo. Cuando un agente viaja a través de la red, ellos migran de un servidor de agentes a otro.

- *Comunicaciones:* Que facilite la comunicación de los agentes con el mundo exterior, principalmente, que le permitan interactuar con otros agentes en tareas cooperativas.

- *Nombrado:* Que permite nombrar a los agentes de manera que puedan ser identificados de forma unívoca, y también nombrar a las plataformas y nodos a los que migran los agentes. Cada agente debe tener un identificador único que lo diferencie del resto de agentes de una aplicación.

- *Descubrimiento y localización*: Que facilite a los agentes descubrir otros agentes y plataformas con los que puede interactuar, o a las que puede migrar, para alcanzar los objetivos que tiene encomendados.

- *Movilidad:* Que facilite la migración de agentes entre nodos remotos. Cada agente debe contener en su estructura interna un itinerario que le permita registrar los nodos de una red que va a recorrer. [Maes 1991]

- *Seguridad:* Que garantice por una parte la protección de los agentes ante ataques del nodo en el que se ejecuta y por otra, la protección del nodo ante los ataques de los agentes que se ejecuten en el.

- Control y manejo de los agentes: Los agentes de una plataforma de agentes móviles, deben proveer mecanismos de control de sus actividades, como monitoreo de su estado de ejecución y distribución de sus tareas o acciones de ejecución. [Bigus 2002]

Las nuevas plataformas copian las características que ofrecían las anteriores y añaden nuevos conceptos que facilitan el desarrollo de las aplicaciones, la aparición de Java originó un desarrollo más rápido de la tecnología de agentes

incorporando al mercado una gran cantidad de sistemas de agentes móviles basados en Java de los que se presentara las más importantes.

1.4.1. IBM Aglets

Uno de los proyectos de IBM es el Aglets Workbench (AWB), esta es una plataforma escrita en Java [Lawson 1996]. AWB está basado en el concepto de aglet que es un agente móvil basado en Java. Aglets Workbench usa una técnica llamada serialización para transmitir datos y migrar el byte-code interpretable. [Lang 1998]

Los aglets son objetos Java que se pueden mover a través de la red de un computador a otro. Es decir, un aglet que se esté ejecutando en una máquina puede detener su ejecución, desplazarse a otra máquina remota y reiniciar su ejecución de nuevo en dicha máquina. Cuando un aglet se mueve, lleva consigo el código del propio aglet así como el estado de todos los objetos que lo constituyen. Un mecanismo de seguridad interno hace que sea seguro el hospedar aglets no confiables (untrusted).

Aglets tiene un punto de entrada bien definido para reinicio de cómputo y soporta persistencia. Llamando a las funciones apropiadas basadas en clases, se pueden mantener temporalmente los aglets en almacenamientos secundarios y activarlos después. [Perez 2000]

Los componentes de Aglets Workbench son: Un ambiente gráfico (o visualizador de aglets) para la construcción de aplicaciones de agentes móviles en Java conocido como Tahiti, un servidor de agentes, la especificación de un Protocolo de Transferencia de Agentes (ATP) y finalmente los mismos aglets. El visualizador de aglets es el equivalente al Appletviewer utilizado para applets en Java.

En el servidor de Aglets se pueden crear, activar, retractar, desactivar o "despachar" aglets. Los servidores de Aglets son máquinas que pueden recibir gran cantidad de aglets y con gran cantidad de datos o recursos de cómputo. Los aglets viven en el contexto de servidores de Aglets. Estos servidores refuerzan su política de seguridad configurando su manejador de seguridad.

Cuando un aglet es despachado acarrea su itinerario que es la lista de lugares que posteriormente visitará en el proceso de migración. Los Aglets durante su tiempo de vida, visitarán servidores de Aglets, ejecutarán tareas y finalmente acarrearán un resultado de regreso. La característica clave cuando se considera tecnología

de agentes de red es la movilidad. Esto implica que los agentes pueden moverse en la red de acuerdo a algún itinerario, es decir, la ruta que el aglet seguirá cuando es instanciado.

La estructura de Aglets consiste de dos partes, el núcleo del aglet y el proxy del Aglet, vistos en la figura 3. El núcleo es el corazón del aglet y contiene todas las variables internas del aglet y los métodos. Proporciona interfaces a través de las cuales el aglet puede comunicarse con su medio ambiente. El núcleo del Aglet es entonces encapsulado por el proxy del aglet que actúa como un protector contra el acceso directo a cualquier método y variables privadas además puede ocultar la posición real del aglet.

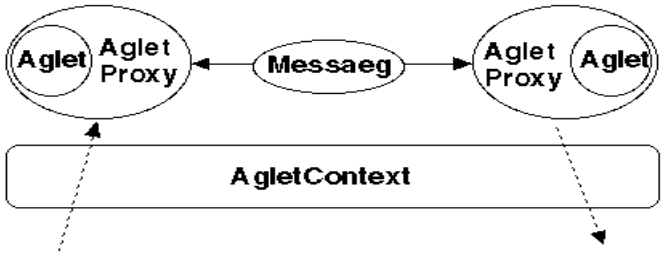

Ilustración 3. ***Estructura de Aglets***

Un aglet también contiene un identificador único y un itinerario, los aglets no pueden migrar a sistemas donde no exista un servidor compatible, en el cual ellos puedan ser ejecutados. Estos servidores, conocidos como contextos, actúan como las casas o lugares de trabajo donde los aglets de todo tipo pueden comunicarse unos con otros.

Un contexto es un objeto estacionario que proporciona un medio de recibir y manejar aglets en un ambiente seguro para su ejeucion. [Shoham 1993]

A través de este contexto (figura 3) un aglet es capaz de conseguir información acerca de su ambiente, y enviar y recibir mensajes a ese medio y a otros aglets activos en su proximidad. Si un aglet da o intenta violar el proxy aglet y accesar datos privados, el manejador de seguridad detectará el intento y detendrá el acceso. Los aglets pueden comunicarse entre ellos pasando mensajes a través del contexto. Este proceso entabla el intercambio de un objeto mensaje entre dos aglets, y permite el paso de mensajes síncronos y asíncronos, así que habilita a los aglets para colaborar e intercambiar información de una manera estricta y relajada. [Lang 1998]

Aglets Workbench soporta movilidad e itinerario. La movilidad en Aglets Workbench es posible gracias a dos facilidades:

1. El Protocolo de Transferencia de Agentes (ATP).
2. La Interfaz Java de Transferencia y Comunicación de Agentes (J-ATCI).

El ATP es una aplicación al nivel de protocolo para agentes distribuidos basado en sistemas de información que facilita la migración de los aglets sobre la red. Basado en las convenciones de Internet, ATP usa el Universal Resource Locator (URL) para especificar direcciones de los nodos, mantiene una plataforma de protocolo independiente para habilitar la transferencia de agentes móviles entre computadores de una red. Aunque este protocolo ha sido liberado con Aglets Workbench su dominio no es de uso exclusivo para aglets, ofrece la oportunidad de manejar agentes móviles desde cualquier lenguaje de programación y una variedad de sistemas de agentes. [Lang 1998].

La plataforma de Aglets Workbench no soporta la migración de sus agentes entre dispositivos móviles, solo existe la movilidad entre dispositivos fijos de una red.

1.4.2. Telescript

La primera implementación comercial del concepto de agentes móviles, fue la tecnología Telescript de General Magic[10], esta tecnología intentó permitir acceso automático e interactivo a las redes de computadores usando agentes móviles. El enfoque comercial de la tecnología de General Magic fue el comercio electrónico, que requería que una red permitiera a los proveedores y consumidores de servicios encontrarse uno al otro y hacer negocios electrónicamente. Los creadores de Telescript apreciaron al comercio electrónico como una pequeña pieza del mundo de agentes móviles que existirá en los próximos años.

Telescript implementa una arquitectura que abarca diversos módulos y conceptos, que le permiten crear sistemas asociados con programación remota. La arquitectura incluye una división de módulos que favorecen su objetivo principal, el comercio electrónico, así pues se incluyen lugares, puntos de reunión, autoridades, agentes, conexiones y permisos que son necesarios para que un sistema de agentes móviles pueda resolver dicha tarea. Para facilitar el desarrollo de aplicaciones de comunicación, los agentes y los lugares son escritos en el lenguaje de Telescrip, el cual posee las siguientes características:

- Es un lenguaje completo.

[10] General Magic: Referirse a [Tomas 1998]

- Orientado a objetos.
- Dinámico.
- Persistente.
- Portable y seguro.

Además del lenguaje de programación, Telescript provee un servidor (llamado máquina en el lenguaje de Telescript) que es un programa de software que implementa el lenguaje para mantener y ejecutar los lugares dentro de su contexto, al igual que a los agentes que ocupan esos lugares. El lenguaje ofrecía la mayor parte de los requerimientos que requería un agente en el comercio electrónico: seguridad, movilidad y transacciones. Telescript y sus tecnologías asociadas son ampliamente reconocidas como los fundadores de las bases, filosófica y técnica, para la mayoría de los otros sistemas de agentes móviles que se usan hoy en día.

La plataforma de Telescript fue precursora de los sistemas de agentes móviles, quizá su único problema por el que no tuvo éxito, fue su incompatibilidad con algún lenguaje de programación existente, lo que implicaba que los usuarios tuvieran que aprender un nuevo lenguaje de programación para desarrollar sistemas de agentes. Aunado a esto debemos añadir una mala campaña de mercadotecnia, una desventaja que se encontró a futuro es que no da soporte a la migración de agentes en dispositivos móviles. [Perez 2000].

1.4.3. ARA (Agentes Para Acceso Remoto)

ARA es un proyecto del área de sistemas distribuidos del departamento de informática de la Universidad de Kaiserslautern [KAISa], Alemania. La idea básica del sistema de agentes para acceso remoto ARA (de la lengua inglesa Agents for Remote Access) es crear una plataforma capaz de moverse libre y fácilmente sin interferir con su ejecución, utilizando diferentes lenguajes de programación existentes, independiente de los sistemas operativos y de las máquinas que participan. El sistema proporciona facilidades para los requerimientos específicos de los agentes móviles en aplicaciones reales, en donde la seguridad es importante.

Ara es un buen ejemplo de software intermedio, situado entre las aplicaciones específicas y el sistema operativo que hay debajo. Proporciona las facilidades al nivel de sistema para ejecutar y mover programas, permitiéndoles interactuar y acceder a su sistema servidor, todo ello de una forma portable y segura .

Generalidades de Ara

ARA consiste en un grupo de agentes moviéndose entre lugares, donde ellos usan ciertos servicios para hacer su trabajo.

Los agentes móviles en ARA son programados en un lenguaje interpretado y ejecutados dentro de un intérprete para este el lenguaje, usando un runtime especial para los agentes, llamado el núcleo en términos de ARA. Sin embargo la relación entre el núcleo y el intérprete es característico de ARA: aislar los temas específicos del lenguaje en el intérprete, mientras se concentra toda la funcionalidad independiente del lenguaje en el núcleo. Esta separación de intereses hace posible utilizar diversos intérpretes a la vez para diferentes lenguajes de programación sobre un núcleo común y genérico [Lingau 1996].

El núcleo trabaja sólo con los agentes en general, haciendo sus servicios disponibles uniformemente a todos los agentes sin importar sus respectivos lenguajes interpretados. El esquema completo de ARA se simplifica en: intérpretes y núcleo corriendo como un proceso de aplicación simple sobre un sistema operativo servidor sin modificaciones. Actualmente, los intérpretes para los lenguajes de programación Tcl y C han sido adaptados en el núcleo de Ara, y Java será añadido próximamente.

La arquitectura de ARA

Los agentes se ejecutan dentro de interpretes para su respectivos lenguajes de programación, controlados y servidos por el núcleo de un sistema común. Éste es el principio fundamental de ARA, ejecutar los agentes como procesos concurrentes y autónomos. Esta idea soporta su independencia y posiblemente la ejecución asíncrona, provee control flexible y facilita la protección mutua.

El núcleo de ARA visto en la figura 6 proporciona su propia abstracción de procesos como la base para la implementación de agentes. Existe usualmente un intérprete ejecutándose, el cual procesa el programa de un agente móvil. El núcleo gobierna los procesos del agente, y realiza la mediación de su interacción y el acceso al servidor del sistema. Las funciones básicas, como migración, son proporcionadas a los agentes por el núcleo, mientras que los servicios de alto nivel son ofrecidos por agentes servidores.

Los agentes estacionarios pueden ser compilados ya que éstos serán locales y contribuirán con el núcleo en la gestión del sistema [Lange 1998].

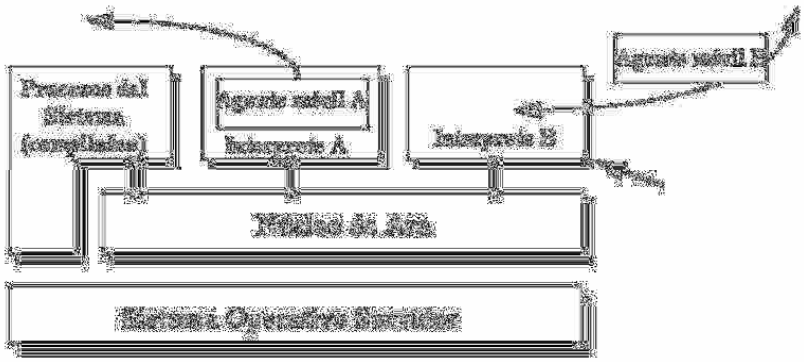

Ilustración 4 Arquitectura de ARA

El sistema ARA también emplea procesos para ciertos propósitos internos, con el fin de modularizar la arquitectura. Los procesos del sistema tienen permisos especiales y pueden directamente acceder al sistema operativo servidor, pasando por el núcleo.

Como la implementación de procesos es interna en el sistema de ARA, el ensamblaje completo de agentes, intérpretes y núcleo se ejecutará como un proceso simple de aplicación en la cima de un sistema operativo servidor intacto, como se puede apreciar en la figura 6 lo que facilita considerablemente la portabilidad a plataformas específicas. Cabe resaltar que la migración que soporta la plataforma ARA no brinda los medios para mover agentes entre dispositivos móviles.

1.4.4. Agent TCL

Agent TCL es un poderoso sistema de agentes que se ejecuta en estaciones de trabajo UNIX. Agent TCL es una plataforma efectiva para la experimentación y el desarrollo de aplicaciones de pequeño y mediano alcance.

Los agentes de Agent TCL están desarrollados en una versión extendida del lenguaje de comandos TCL (Tool Command Lenguage) desarrollado por Sun Microsystems. TCL es un lenguaje de codificación de alto nivel y es a la vez poderoso y fácil de aprender. Los agentes de Agent TCL pueden usar todos los comandos estándar de TCL así como un conjunto de comandos especiales que

son provistos cómo una extensión a TCL. Estos comandos especiales permiten a un agente migrar de una máquina a otra, crear agentes hijos, comunicarse con otros agentes, así como obtener información acerca de la localización actual en la red de los agentes. Adicionalmente, Agent TCL puede ser extendido con un conjunto de comandos definidos por el usuario para crear un sistema de agentes más poderoso.

El nombre de Agent TCL fue modificado a D'Agents en la última versión del producto, al considerar la modificación de su núcleo para que el producto ofreciera la capacidad de soportar diversos lenguajes para el desarrollo de agentes (Tcl, Java y Python). De aquí que sus autores consideraran que ya no debería llamarse Agent TCL, y hayan decidido un nombre más genérico es decir: D'Agents. [GRAY 1997]

Arquitectura de D'Agents

Esta arquitectura está construida sobre el modelo servidor de Telescript, los lenguajes múltiples de ARA, y el mecanismo de transporte de dos sistemas predecesores en Dartmouth. La arquitectura consta de cuatro niveles, obsérvese la figura 4. El nivel más bajo es una interfaz de programación de aplicaciones(API) para cada mecanismo de transporte disponible.

El segundo nivel es un servidor que se ejecutará en cada nodo de la red a donde los agentes quieran ser enviados. El servidor realiza las siguientes tareas:

Estado: El servidor mantiene un registro de los agentes que están ejecutándose en esa máquina y contesta preguntas acerca de su estado.

Migración : El servidor acepta a cada mensaje que llega, autentica la identidad del propietario, y pasa el agente autenticado al intérprete apropiado para su ejecución. La migración de los agentes solo es posible entre dispositivos fijos (computadores) de una red.

Comunicación: El servidor provee un espacio de nombres jerárquico para los agentes para permitir que los agentes se envíen mensajes unos a otros dentro de ese espacio de nombres. La división más alta del espacio de nombres es el nombre simbólico de localización de la red del agente.

Almacenamiento no volátil: El servidor proporciona acceso a un almacenamiento no volátil para que los agentes puedan realizar copias de seguridad de su estado interno cuando lo deseen. El resto de los servicios son proporcionados por los agentes. Tales servicios incluyen descubrimiento de recursos, comunicación de grupos, tolerancia a fallos, control de acceso, y comunicación independiente de la

localización. Sin embargo los agentes de servicios más importantes en este prototipo son los agentes de reenvío y los agentes de gestión de recursos.

Los agentes de reenvío soportan operaciones de desconexión, si un agente no es capaz de migrar a la localización deseada por que la máquina o red fallan, el agente es añadido a una cola o lugar de reenvío dentro de la red. El agente de envío dirige al agente a la localización deseada hasta que le sea posible alcanzarla. Los agentes gestores de recursos, en combinación con el sistema de encriptación de muy buena seguridad PGP (del inglés pretty good privacity) y con los módulos de seguridad de lenguaje específico como Safe-Tcl, guardan el acceso a recursos críticos del sistema. [GRAY 1997]

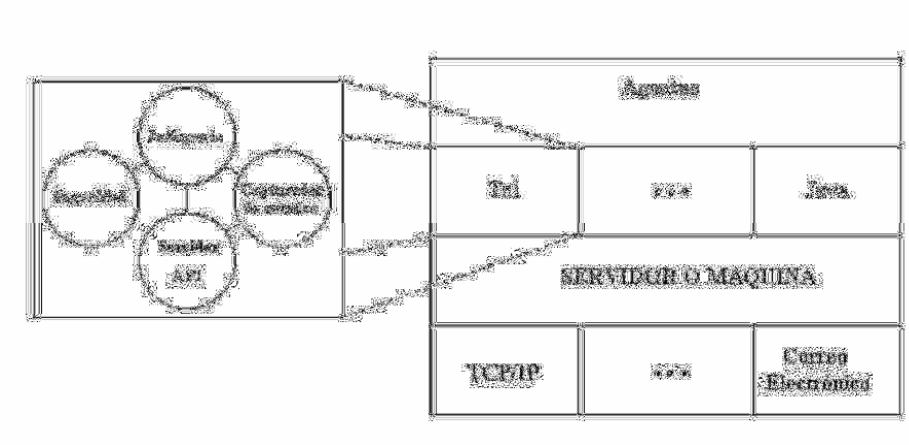

Ilustración 5 Arquitectura D'Agents

La tercera capa de la arquitectura consiste en un intérprete para cada lenguaje disponible. Cada intérprete tiene cuatro componentes: El intérprete mismo, un módulo de seguridad que previene al agente de realizar acciones maliciosas, un modelo de estado que captura y restablece el estado interno de un agente en ejecución, y una API que interacciona con el servidor para gestionar la migración, comunicación y la posición del puntero de ejecución. Al añadir un nuevo lenguaje es necesario crear todos los componentes arriba mencionados para ese lenguaje, excepto el de seguridad, ya que este módulo solo asegura que un agente no pasa la gestión de recursos o viola las restricciones de seguridad impuestas por el gestor.

El nivel más alto de la arquitectura de D'Agents consiste en los mismos agentes. [White 1996].

1.4.5. Jade

Es una plataforma desarrollada en java bajo la filosofía Open Source, para el desarrollo de aplicaciones distribuidas basadas en Agentes y cumple con la especificación FIPA[11] (Foundation for Intelligent Physical Agents). Su esquema de distribución es visto en la figura 5.

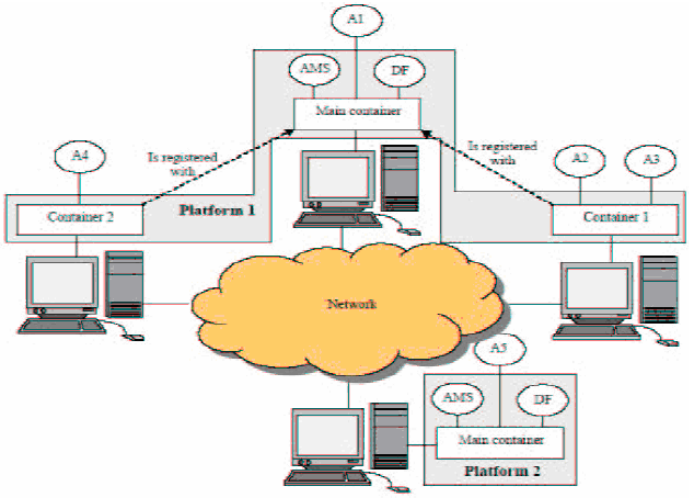

Ilustración 6 Esquema de Distribución de JADE

Contenedor (containers): Entorno de ejecución donde pueden vivir los Agentes JADE.

Plataforma: Conjunto de contenedores que proporciona una capa que oculta a los Agentes y al desarrollador el entorno donde se ha decidido ejecutar la aplicación.

Contenedor Principal: en cada plataforma debe existir este contenedor, y la principal diferencia con los demás contenedores es:

- **AMS (Agent Management System):** este agente se encarga del servicio de nombrado, asegurando que cada agente tenga un nombre único. También tiene el poder de crear y destruir Agentes remotos si la lógica del negocio de determinada aplicación lo requiere.

- **DF (Directory Facilitator):** Proporciona el servicio de páginas amarillas. Gracias a él un Agente puede encontrar a otros agentes.

[11] FIPA: Referirse a [FIPA 2000]

La arquitectura se puede observar en la figura 5, donde aparecen dos plataformas diferentes (platform1 y platform2), cada una con sus contenedores principales y contenedores normales. [MURU 2002]

El problema es que JADE no puede correr en dispositivos móviles debido a diversas razones:

Espacio: La memoria necesaria para todo el ambiente de ejecución de JADE es de varios MB. Los cuales no pueden ser soportados por los dispositivos handheld debido a sus limitaciones.

Versión del JDK: JADE necesita el JDK 1.4 o posterior, mientras que la mayoría de los dispositivos handheld solo soportan Java, CLDC o más típicamente MIDP (JME).

Características propias de las redes inalámbricas: Como son latencia alta (tráfico intenso), IP dinámicas, conectividad intermitente o bajo ancho de banda, hace que JADE no sea lo apropiado.

LEAP (Lightweight Extensible Agent Platform) fue creado para solucionar estos problemas y permite ejecutar agentes JADE en dispositivos handheld y/o conectados a través de redes inalámbricas. Al agregar LEAP, cuando se combina con JADE, hace que se substituyan algunas partes del kernel de JADE formando un ambiente de ejecución modificado identificado como JADE-LEAP y que se puede desplegar en una amplia gama de dispositivos que varían desde servidores hasta teléfonos celulares que soporten java. Para lograr esto, JADE-LEAP puede adaptarse a tres maneras diferentes que corresponden a los tres tipos principales de ambientes Java que pueden encontrarse en los dispositivos que se han considerado:

- JSE: Ejecutar JADE-LEAP en PC y servidores en una red fija corriendo sobre JDK 1.4 o posterior.

- Pjava: Ejecutar JADE-LEAP en dispositivos handheld que soporten CDC de JME o PersonalJava como la mayoría de las PDA´s de hoy en día. Debe anotarse que la especificación PersonalJava fue declarada obsoleta en el 2003 y substituida por la configuración CDC de J2ME, sin embargo desde el punto de vista de funcionamiento de JADE-LEAP no existen diferencias.

- MIDP: Ejecutar JADE-LEAP en dispositivos handheld que soporten MIDP1.0 o posterior, por ejemplo la gran mayoría de teléfonos celulares que soportan java.

Existe una versión de JADE-LEAP para ejecutarse en PC y servidores de red fija que funcionen con la versión 1.1 o posterior de .NET FrameWork de Microsoft. Algunas características que JADE-LEAP tiene disponibles para JSE, pjava y dotnet no son soportados en JADE-LEAP para MIDP. [Habitat 2005].

Los aportes realizados por cada una de las plataformas descritaas anteriomente contribuyeron a aclarar varios conceptos tanto en la arquitectura de la plataforma como en la estructura interna y funcional de los agentes de HERMES. Los aportes o conceptos tomados de estas plataformas son los siguientes:

- De la plataforma JADE de tomo el concepto de manejar dos versiones de la plataforma por problemas de incompatibilidad entre las plataformas base (JSE y JME).
- El concepto de entorno[12] se maneja de igual forma como lo hace aglets con el denominado contexto y contenedor como lo maneha JADE.
- El mantener un núcleo con los datos basicos del agente es un concepto tomado de la plataforma ARA. Quien encapsula los atributos primarios de un agente en su núcleo al igual que HERMES.
- El concepto de Proxy como representante de un agente ante un SMA es tomado de la plataforma Aglets, quien desarrolla este concepto de igual forma que los Proxy de HERMES, pero le da mayor cantidad de responsabilidades a su representante. Asi como el concepto de Itinerario y URL para diferenciar cada sitio registrado en el itinerario que es un registro de los sitios o nodos que visitara el agente en el proceso de migración.
- El concepto de migración de agentes es desarrollado de forma similar que las plataformas mencionadas anteriormente, es decir existe un cliente para enviar agente y un servidor que lo recibe en una maquina remota. La diferencia con el resto de plataformas es la forma de implementar la serialización y el proceso de envio y recepcion de agentes, que debe hacerce para dispositivos móviles, cosa que las anteriores plataformas carecen de este servicio.
- El mecanismo de comunicación difiere del resto de plataforma por que carece de la utilización de un esquema de ACL, por razones propias a la migración, ya que es necesario enviar y recibir objetos propios de HERMES pero que ACL por su composición y funcionamiento interno impide enviar este tipo de objetos.

[12] Entorno: Ambiente de ejecución de los agentes de HERMES.

CAPITULO II

ESTADO DEL ARTE

El desarrollo de proyectos que implementen agentes móviles orientados a la ejecución de los mismos en dispositivos móviles, es un tema que ofrece numerosas posibilidades a los investigadores en el campo académico y empresarial. Por lo tanto los proyectos encaminados hacia la implementación de sistemas multiagentes en redes y dispositivos inalámbricos son muchos, en el presente capítulo se describen algunos proyectos relacionados con el tema de agentes móviles aplicados a dispositivos móviles. Proyectos que en algunos puntos tanto teóricos como de implementación coinciden con el proyecto HERMES, pero no buscan los objetivos primarios del proyecto en mención.

2.1. Aura of Carnegie Mellon University

El proyecto Aura comenzó en el año 2000 y tiene como objetivo crear una arquitectura software de computación ubicua que involucre los conceptos de agentes en sus aplicaciones. El aporte básico de Aura es que su interacción con la tecnología embebida en el entorno sea mínima, de manera que no lo distraiga de sus tareas habituales. Aura se ha desarrollado para un entorno ubicuo en el que interaccionan dispositivos personales, dispositivos embebidos y dispositivos móviles empleando protocolos de comunicación inalámbricos.

En Aura se trabaja con dos conceptos fundamentales:

- El primero se denomina proactivity, que es la capacidad que tienen las capas del sistema donde están implementados los agentes para anticiparse a las peticiones realizadas a alto nivel.

- El segundo se denomina self-tuning, que consiste en que los agentes se adaptan al consumo de recursos y rendimiento a las peticiones y uso que se están haciendo de ellas.

Teniendo como base estos conceptos han definido una arquitectura para dispositivos limitados con las siguientes capas:

- Kernel: basado en Linux.

- Odyssey: Que permite controlar recursos y da soporte a la adaptación de las aplicaciones al contexto.

- Coda: Que proporciona acceso a los archivos adaptados a las diferentes condiciones de ancho de banda de la conexión inalámbrica, también a la movilidad de los dispositivos permitiendo una operación desconectada.

- Spectra: Que proporciona un mecanismo de ejecución remoto que emplea el contexto del usuario para decidir cual es la mejor forma de ejecutar la petición remota realizada.

- Prism: Que permite capturar y gestionar las intenciones de los usuarios y es una capa por encima de la de aplicación.

En Aura para aumentar las capacidades de los dispositivos limitados se emplea lo que se denomina cyber foraign que son típicamente computadores de escritorio que se embeben en los entornos por los que se mueve el usuario y que los dispositivos limitados emplean para delegar ciertas tareas en ellos y a través de los que se conectan a Internet.

Dentro del proyecto Aura se han realizado también desarrollos orientados a lo que se denominan advisor devices para proporcionar información de las condiciones de la red, y a lo que se denomina people locator que permiten ofrecer servicios teniendo en cuenta la localización y el contexto del usuario. Ambos trabajos se han realizado para el protocolo inalámbrico IEEE 802.11. [Qusay 2001].

La relacion que existe entre el proyecto Aura y HERMES radica en el implementa agentes para dispositivos inalambricos es decir hay un soporte de agentes pero en sistemas operativos embebidos. Para ambientes ubicuos, en el caso de HERMES sus agentes han sido pensados para dispositivos moviles como celulares, pocket Pc, palm, PDÁ´s. es decir existen agentes pero no móviles y no manipulables al desarrollador ni al usuario final.

2.2. SAHARA y el prototipo MILENIO

Proyecto desarrollado por el departamento de informática de la universidad de Oviedo (España), la cual propone una arquitectura de seguridad integral para los sistemas de agentes móviles (SAHARA) que ofrezca un entorno de ejecución seguro en todo sistema que la implemente y que solucione en gran medida el problema de seguridad que se le ha imputado a la tecnología.

La originalidad del trabajo radica en crear diversas técnicas de protección contra todos los ataques que un sistema de agentes móviles puede sufrir, logrando una solución integral a los problemas de seguridad que tienen los sistemas de agentes móviles actuales. El proyecto describe detalladamente las medidas necesarias para proteger al servidor de agentes contra ataques de agentes móviles y de servidores remotos, especifica las medidas necesarias para proteger la transmisión entre servidores de agentes así como para proteger al agente móvil contra ataques de servidores maliciosos. Es importante resaltar sobre todo la técnica para proteger el estado de datos del agente móvil durante su itinerario.

Para poder implementar la arquitectura de SAHARA se desarrollo conjuntamente un sistema de agentes disponible sobre el cual se pueda implantar. El cual fue denominado MILENIO como sistema de agentes prototipo cuenta con los elementos básicos para que un sistema de agentes pueda funcionar, carecerá de entornos de desarrollo sofisticados con el que ya cuentan algunos sistemas de agentes móviles así como utilidades que son en ocasiones innecesarias como la posibilidad de envío y recepción de mensajes por parte de los agentes.

MILENIO consta de tres capas principales vistas en la figura 7. En la capa superior que corresponde a la capa de aplicación se encuentran todos los agentes definidos por el usuario. La siguiente capa corresponde a la API y a la implementación del sistema de agentes móviles MILENIO que se encarga de ejecutar y administrar a todos los agentes activos en el sistema. [Perez 2000].

SAHARA a diferencia de HERMES es un proyecto que tiene como objetivo principal el desarrollar un mecanismo de seguridad para el transporte de agents en redes inalambricas, despreocupandose de sus propios agents, estructura interna, y funcionalidades, con el proyecto MILENIO buscan dar soporte a un sistema de agente pero solo es usado como herramienta extra para comprobar los mecanismo de seguridad, de este modo difiere de HERMES en su busqueda por obtener un mecanismo optimo para la migración de gentes.

Ilustración 7. Arquitectura de Milenio

2.3. TAgentesP

Es un proyecto realizado por la universidad Carlos III de Madrid (España). Consiste en la realización de una plataforma de agentes distribuida entre el SIM Java Card y dispositivos móviles soportados por la plataforma JME, teniendo en cuenta que la parte ligada a la personalización de servicios debe residir en la tarjeta para que se mantenga la característica de movilidad del usuario y acceso global a los servicios independiente de la localización del dispositivo móvil y de la red de acceso. En la figura 8 se observa un esquema de la arquitectura propuesta, por los autores del proyecto.

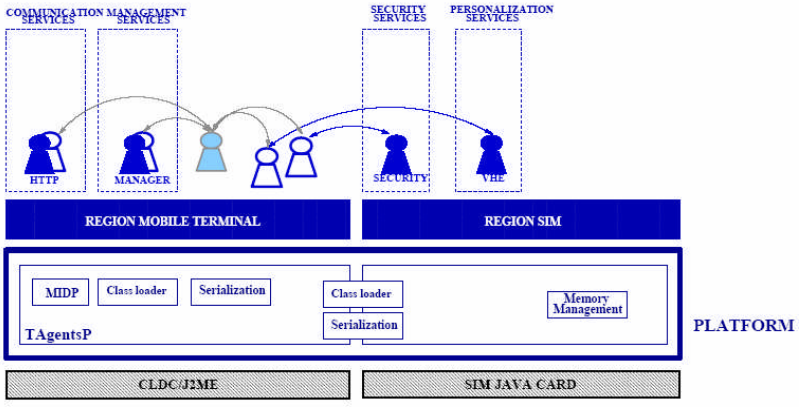

Ilustración 8. Arquitectura de la Plataforma de Agentes Móviles

El desarrollo de la plataforma 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 se le ha llamado TAgentsP (Travel Agents Profile).

Las limitaciones del lenguaje Java en su versión JME obligan a abordar dos problemas importantes para una plataforma de agentes y que en la versión JSE resolvía el propio lenguaje. Por una parte es necesario dotar de clases que permitan la serialización de objetos y por otra permitir la carga dinámica de clases.

Una vez proporcionados los servicios básicos, se provee a la plataforma de componentes que permitan alojar a los agentes del sistema para proporcionar servicios de comunicación tanto para la migración, como para la comunicación de los agentes residentes en otros terminales móviles o en sistemas de la parte fija de la red. En su primera versión, la comunicación se realiza empleando el protocolo HTTP, proporcionado por el perfil MIDP estándar. En versiones posteriores se completa permitiendo la comunicación a través de sockets.

Además se proporciona servicios de gestión de la plataforma que permita manejar los agentes residentes tanto en la región SIM como en el dispositivo móvil.

Los agentes que se ejecuten sobre la plataforma en mención proporcionarán servicios añadidos al usuario móvil (aplicaciones de comercio electrónico, servicios basados en localización, juegos) pertenecientes al propio operador móvil o a terceros proveedores de aplicaciones.

Plataforma sobre el SIM Java Card.

Las limitaciones de Java Card llevan a importantes retos a la hora de proporcionar servicios básicos de una plataforma de agentes. La primera tarea que se busco resolver fue implementar un gestor de memoria dinámico debido a la falta de recolector de basura en Java Card. Se implemento entonces mecanismos que permitan la transferencia de agentes, para ello fue necesario proporcionar mecanismos de serialización y de carga dinámica de clases. Por las limitaciones de Java Card y el mecanismo de comunicación de las tarjetas inteligentes, la realización de estas tareas deberá distribuirse entre la propia tarjeta y del dispositivo móvil.

Los componentes desarrollados en la SIM proporcionan los servicios de seguridad y de personalización a través de agentes del sistema. Los servicios de seguridad permiten añadir credenciales a los agentes que se ejecuten sobre la plataforma y

que vayan a migrar a otros sistemas y además permiten realizar comprobaciones de credenciales de agentes que visiten la plataforma.

Los servicios de personalización, proporcionados por el agente del sistema denominado VHE, permiten personalizar agentes según las preferencias del usuario, según su localización y según sus características de suscripción a determinados servicios, además permiten indicar a los agentes que deseen migrar a la plataforma las características del usuario para que puedan personalizar y adaptar los servicios ofrecidos.

El acceso a estos agentes desde el dispositivo móvil se realizará a través de clones de los agentes del sistema, que residirán en el dispositivo móvil y que permitirán el acceso seguro a los servicios proporcionados.

Debido a la información que contiene el SIM, se desarrollo un componente en la plataforma sólo para ejecutar agentes pertenecientes al operador móvil al que está suscrito el usuario. Estos agentes posiblemente estarán dedicados a tareas destinadas a la gestión de red y actualización de información de suscripción del usuario [Garcia 2002].

La implementación de migración que ofrece TAgentsP a travez de su plataforma sobre la sim java card sugiere tener una versión exclusiva de JME ademas de que la migración depende de la sim, cosa que difiere de la migración que brinda HERMES donde se utiiza una version comercial de JME y la migración es tarea exclusiva de l plataforma y no depende de una sim.

2.4. MAE of Monash University

En Monash University 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[13]. MAE está compuesta por un device agent execution environment (DAEE) que proporciona un conjunto de servicios a los agentes que se ejecutan en la plataforma.

Estos servicios se agrupan en tres tipos: presentación, que proporciona los servicios que permiten a los agentes interaccionar con el usuario; agent, que proporciona los servicios que permiten a los agentes interactuar con los servicios de red, entre ellos se encuentra el que permite descubrir y anunciar servicios y el

[13] JNI: Interfaz de Java nativo

que permite conocer el estado de la red; network, que permite a los agentes comunicarse con otros dispositivos.

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 JME [Celeste 2004]

MAE implemento la migración de sus agentes exclusivamente para sistemas basados en PalmOS, pero el manejo en cuanto a la estructura funcional de los agentes es similar a HERMES por que mantiene un sietema de nombrado de los agentes implementa un mecanismo de comunicación y mantien una estructura interna de los agentes para sus propositos específicos.

2.5. Working Group FIPA Ad Hoc

El Working Group FIPA Ad Hoc surge a principios de 2002 con el objetivo de definir una plataforma de agentes conforme con FIPA, que pueda operar en redes ad-hoc. El principal problema al que se enfrentan en este tipo de redes, es que son redes sin infraestructura y entonces, la plataforma existente en cada dispositivo debe ser autónoma, y por lo tanto poseer todos los componentes definidos en el modelo de referencia de FIPA.

El tema mas activo del grupo de desarrollo se centra en determinar como debe realizarse el descubrimiento de servicios/agentes y por lo tanto, analizan si la definición de Directory Facilitator y los mecanismos de federación para búsquedas remotas en FIPA es valido y posible en redes ad-hoc.

El grupo de desarrollo fundamenta sus trabajos en tres requisitos:

- Añadir los mínimos cambios a las especificaciones existentes de FIPA.
- Utilizar los mecanismos de descubrimiento dinámico de servicios existentes en la actualidad (Jini, JXTA, SLP, SSDP, etc), de manera que la solución propuesta pueda ínteroperar con cualquiera de ellos.
- Adaptarse a entornos ad-hoc, pero que la solución propuesta también sea valida para redes con infraestructura [WG-AdHoc 2002].

La diferencia radical del proyecto Working Group FIPA Ad Hoc con HERMES es que la migración de sus agentes se centra esxlusivamente en redes ad Hoc.

CAPITULO III

DESCRIPCION DE LA PLATAFORMA HERMES

3.1. Arquitectura del Ambiente

La arquitectura del ambiente es la estructura funcional que permite implementar y ejecutar una aplicación multiAgente con la plataforma HERMES.

La arquitectura del ambiente apreciada en la figura 9, muestra una vista general del sistema HERMES donde se puede observar cada uno de sus componentes y las relaciones que existen entre ellos. La arquitectura del ambiente muestra los diferentes niveles en que se divide la plataforma HERMES

El proyecto HERMES desarrolló dos versiones de la plataforma, la versión para dispositivos fijos, que es soportada por JSE y la versión para dispositivos móviles soportada por JME.

La versión para dispositivos fijos se basa en la API de la plataforma J2SE, los agentes que se implementen en la versión para dispositivos fijos de HERMES se deben diseñar e implementar teniendo en cuenta su plataforma base.

La versión para dispositivos móviles se soporta en la plataforma JME, una versión muy reducida de la plataforma JSE. Debe tenerse en cuenta que los agentes que se implementan bajo la versión para dispositivos móviles, tendrán una menor flexibilidad en cuanto a servicios prestados ya que su API no cuenta con tantos servicios como la versión soporta por JSE.

A continuación se describe cada uno de los niveles de la arquitectura del ambiente de HERMES.

El *nivel 3* representa las plataformas JSE y JME. Estas plataformas tienen una API sobre las que se sustentan las dos versiones de HERMES, una versión para equipos fijos y otra para dispositivos móviles.

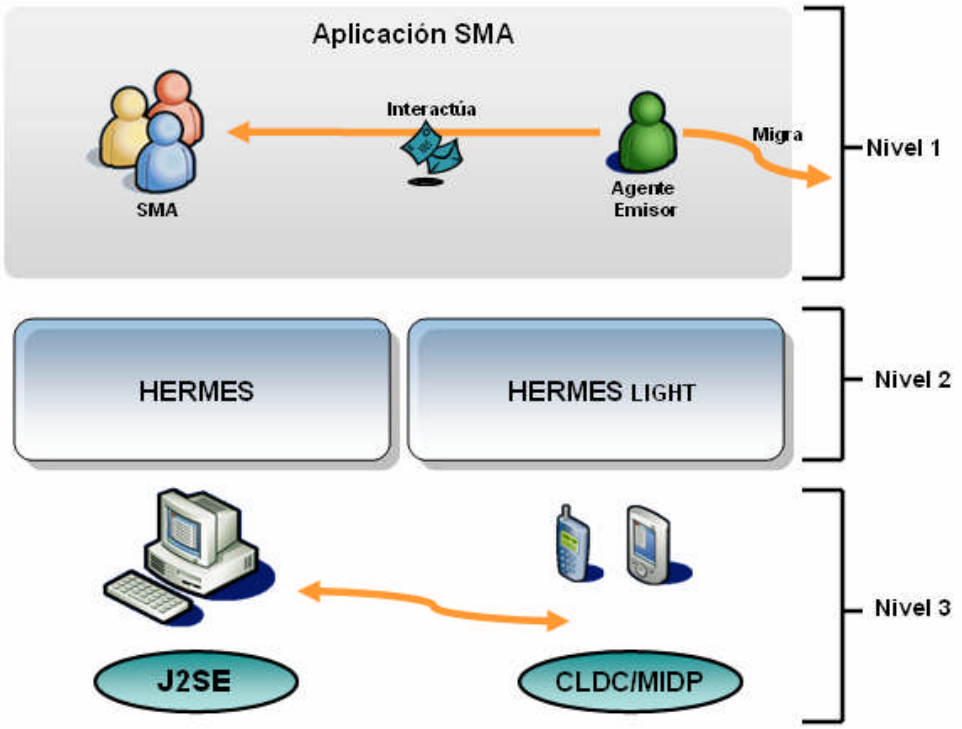

Ilustración 9 Arquitectura del Ambiente.

El *nivel 2* representa las dos versiones de la plataforma HERMES, la versión para dispositivos móviles (HERMES Light) y la versión para dispositivos fijos (HERMES).

En el *nivel 1* de la arquitectura, se muestra como esta distribuida cualquier tipo de aplicación implementada con el API de HERMES dicha distribución consiste en que independiente de la versión de HERMES, la migración y ejecución de los agentes podra implementarse en cualquiera de los dispositivos móvil o fijo. Ademas de que en la aplicación MultiAgente se puede crear, eliminar y comunicar agentes dependiendo de la lógica que el desarrollador necesite implementar.

3.2. Arquitectura de los Agentes HERMES

Un agente HERMES está constituido en su estructura interna por cuatro componentes: identificador, estado, itinerario y comportamiento. Estos componentes determinan características en los agentes, tales como:

- Los cambios de estado de un agente a lo largo de su ciclo de vida.
- El modo de actuar y reaccionar del agente ante eventos generados en su ambiente de ejecución.
- Otorga una identificación a los agentes y clon[14]es de la plataforma HERMES.
- Ofrecer mecanismos de control y gestión para manipular los viajes de los agentes.

La composición interna del agente se muestra en la figura 10.

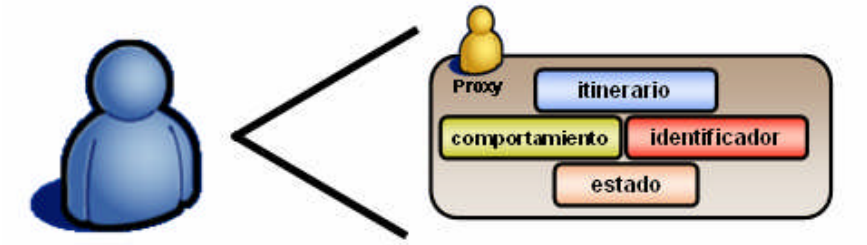

Ilustración 10 Composición interna del agente

Proxy

Componente que representa al agente en la aplicación multiAgente, entre sus características más relevantes están:

- Encapsula la estructura interna del agente.
- El acceso a los componentes de la estructura interna se hace a través del Proxy.

[14] Clones: Es una copia de un agente con caracteristicas y comportamientos similares al agente padre.

- Actúa como un intermediario entre el agente y el SMA en los procesos de comunicación.

La labor del Proxy es transparente al desarrollador, su manipulación es gestionada por la plataforma HERMES.

Identificador.

Componente que otorga al agente una identidad mediante el uso de un nombre lógico. El nombre lógico permite diferenciar agentes y clones. El identificador es necesario en los procesos de validación y búsqueda de los registros de una aplicación Multiagente. El manejo de un sistema de nombrado para los agentes es tarea del sistema de localización quien utiliza el identificador y otros elementos como el entorno para componer una estructura de nombrado para los agentes de la plataforma HERMES.

La asignación del nombre lógico es responsabilidad del desarrollador de aplicaciones. Durante el proceso de creación de agentes se realiza la operación de asignación y registro de la identidad del agente. Existe un mecanismo de seguridad en la plataforma HERMES que impide la ambigüedad en la conformación del nombre lógico para agentes y clones.

Itinerario.

Componente que permite gestionar, controlar y manipular la forma en que se realizan los viajes de los agentes. El itinerario hace uso de una URL como un elemento que especifica el destino donde se presume se realizarán los viajes del agente. Este elemento es almacenado en una lista de destinos gestionados por el Itinerario.

Cada URL registrada en el itinerario representa la máquina fija o móvil donde migra el agente. Este elemento tiene una composición interna vista en la figura 11, que facilita los procesos de migración, búsqueda y localización de un destino.

Ilustración 11. Composición interna de la URL.

Cada elemento que integra la composición interna de la URL debe ser inicializado por el desarrollador de aplicaciones cuando adiciona una URL al Itinerario. A continuación se describen los elementos que componen la estructura interna de la URL.

Nombre Lógico: Este valor identifica el destino donde migrará el agente.

Dirección IP Origen: Dirección IP real de la máquina de donde parte el agente.

Dirección IP Destino: Dirección IP real de la máquina a donde viajará el agente.

Entorno Origen: Nombre lógico del entorno de donde parte el agente.

Entorno Destino: Nombre lógico del entorno a donde viajará el agente.

Puerto Origen: Puerto por el cual escucha el servidor de donde parte el agente. Si este elemento no se fija se usará el puerto 4000.

Puerto Destino: Puerto por el cual escucha el servidor a donde viajará el agente. Si este elemento no se fija se usará el puerto 4000.

El itinerario ofrece al desarrollador de aplicaciones manipular la lista de destinos a través de mecanismos de control que permiten:

- Conformar el orden en que será visitado cada destino registrado en el itinerario.
- Adicionar nuevos destinos.
- Actualizar la lista de destinos donde viajará el agente y eliminar los destinos que ya no se requieran en el proceso de migración.

Tener varios destinos donde migrar a disposición del agente, permite contar con más de una alternativa para escoger el destino de ejecución de una tarea o del agente.

Estado.

Componente que representa cada uno de los cambios por los cuales pasa un agente durante su ciclo de vida. Este componente es el encargado de controlar los

cambios ocurridos en el agente y conservar el valor de cada uno de ellos. Su tarea Comienza desde la creación del agente hasta su posterior destrucción.

Una vez el agente es creado, tiene un estado inicial, a menos que el desarrollador de aplicaciones determine lo contrario. Este componente tiene una serie de estados que pueden cambiar por la percepción del ambiente y los comportamientos del mismo, hasta llegar a un estado final una vez terminado su ciclo de vida. Los estados por los cuales puede pasar un agente son descritos en la siguiente figura:

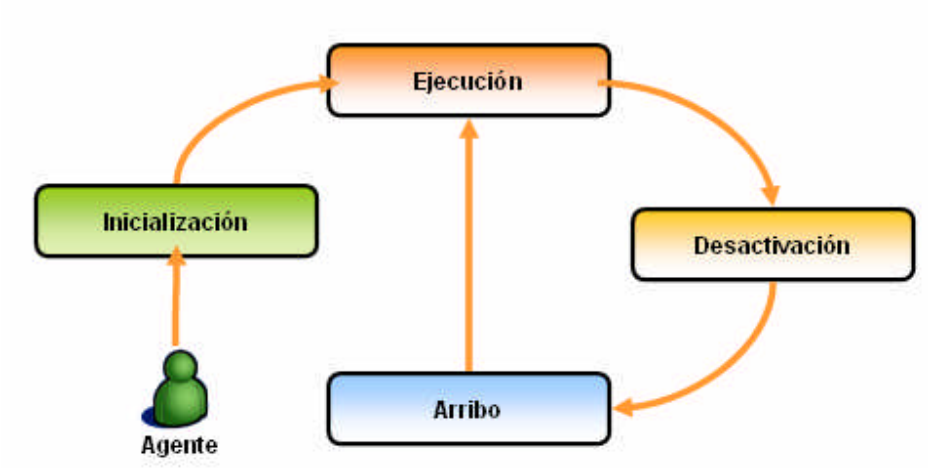

Ilustración 12. Diagrama de Estados del Agente HERMES.

1. Inicialización: Estado que representa el valor inicial que adopta el agente una vez creado. Este estado es asignado por defecto al agente en el proceso de creación.

2. Ejecución: Valor que activa la ejecución de las tareas de un agente.

3. Desactivación: Estado que suspende la ejecución de las tareas del agente ya sea después de la ejecución de una tarea o después del paso de algún otro estado.

4. Arribo: Este estado representa la acción de llegada de un agente a un entorno específico. El valor que toma el estado de arribó está determinado por los eventos de despacho y recepción del agente en el proceso de migración y conserva el valor que indica si el agente partió o llegó a su destino.

Comportamiento.

Componente que permite agrupar las tareas[15] del agente, su registro es gestionado por el mismo agente, de modo que su ejecución se realice en bloque. Controla la ejecución de las tareas de acuerdo al orden fijado por el desarrollador.

Existen tres atributos que maneja el comportamiento y son establecidos por el desarrollador de aplicaciones.

- Nombre lógico: Identifica a cada comportamiento creado y lo diferencia de los demás.

- Prioridad de ejecución: Determina el orden de ejecución del comportamiento.

- Estado de ejecución: Permite conocer si un comportamiento ya ha sido ejecutado o no.

El propósito del comportamiento es adicionar cada una de las tareas creadas para el agente, de modo que las tareas agrupadas en un comportamiento tengan una característica común. Para gestionar el manejo de cada una de las tareas el comportamiento mantiene un listado de las mismas.

Una tarea representa las acciones que el agente va a ejecutar en la aplicación. Cada acción del agente es vista como un procedimiento a nivel de código, un ejemplo sería la consulta a una base de datos, la impresión en pantalla de una operación matemática, la transferencia de un mensaje de un agente a otro, etc.

Cada tarea tiene tres atributos en su estructura interna que deben ser fijados por el desarrollador de aplicaciones, previamente a la adición de tareas a un comportamiento específico. Los atributos de la tarea son necesarios para efectos de registro y control de ejecución de las mismas.

- Nombre lógico: Identifica a cada tarea implementada y la diferencia de las demás.

- Prioridad de ejecución: Determina el orden en que cada tarea es ejecutada en un grupo de tareas.

[15] Tarea: Acción o modo de actuar de un agente con un propósito bien definido

- Estado de ejecución: Permiten conocer si una tarea ya ha sido ejecutada o no.

El éxito en la ejecución de las tareas de un comportamiento determinan si el propósito del mismo se cumplió o no. Las tareas manipulan la información del agente y de la aplicación multiAgente. Su función es ejecutar las labores y procedimientos del agente.

Existe dos modos de definir la forma en que se van a ejecutar los comportamientos:

- *Comportamientos Locales*:

  Agrupan las tareas que se ejecutan en forma local, la ejecución local insinúa que el Agente debe realizar sus tareas en la máquina residente, es decir en donde el agente fue creado. El Agente hará uso de los recursos locales de la máquina donde se esté ejecutando (BD, memoria discos, impresoras, etc.).

- Comportamientos Remotos:

  Agrupan las tareas que se ejecutan en forma remota, este modo de ejecución involucra procesos de migración, sincronización y comunicación entre los distintos entornos y Agentes participes en el proceso de ejecución remota. El diseñador de aplicaciones sabe que instrucciones serán ejecutadas en máquinas remotas y con que fin. De este modo se evita problemas de incompatibilidad en las API de las plataformas móvil y fija de HERMES.

La implementación de las tareas ya sea local o remota, se deja a decisión del programador quien determina donde se llevará a cabo la ejecución de la tarea.

3.3. Arquitectura de la Plataforma HERMES

La arquitectura HERMES está compuesta por tres niveles. En el nivel superior está el entorno, en el nivel intermedio se encuentra el ASH[16] y en el último nivel el sistema de soporte de la plataforma (J2SE, J2ME y sistema operativo).

La figura 13 muestra en detalle el orden de cada uno de estos niveles que componen la arquitectura de la plataforma HERMES.

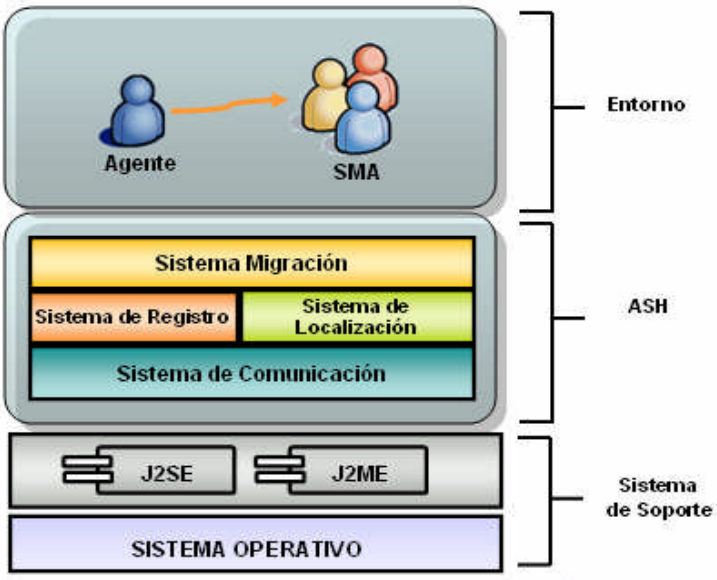

Ilustración 13 Arquitectura de HERMES

Entorno.

El componente denominado Entorno que se muestra en la figura 12 cumple con un propósito específico, agrupar agentes en un sitio determinado. El Entorno ofrece un ambiente de ejecución propicio a los agentes para que desarrollen su ciclo de vida, con las herramientas necesarias para que se lleve a buen término su ejecución.

El concepto de entorno es ampliamente conocido en las plataformas de agentes, para citar algunos casos está la plataforma JADE en la cual este tipo de componente es llamado contenedor y cumple propósitos similares que el entorno en HERMES. De igual forma en la plataforma Aglets existen los contextos que agrupan agentes para proporcionarles ambientes de ejecución óptimos para

[16] ASH: Ambiente de Servicios Hermes

desarrollar sus acciones. Caso muy similar con el propósito de los entornos en HERMES.

La creación de un agente es precedida por la adición de dicho agente a un entorno específico. De este modo el nuevo agente pasa a ser residente del entorno escogido. Los agentes viajeros que llegan a una máquina remota sea fija o móvil pasan a ser registrados en un entorno.

La figura 14 muestra la distribución de los distintos paquetes que conforman el nivel de entorno.

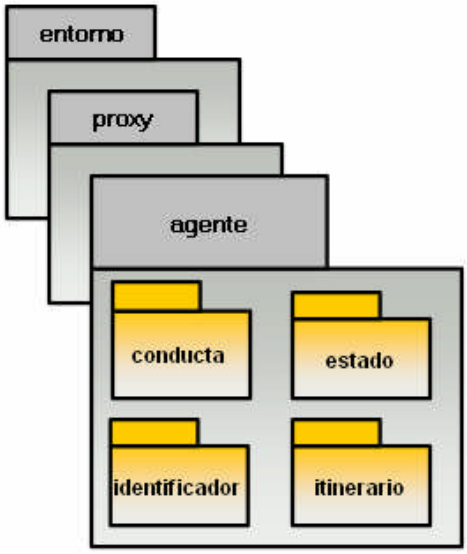

Ilustración 14. Composición de paquetes del nivel de entorno

El paquete entorno: Es la raíz del nivel 1 de la arquitectura y agrupa el Proxy o representante del agente, además ofrece el servicio de la gestión de entornos.

El paquete proxy: Agrupa el núcleo del agente que lo conforman dos elementos, el proxy por el cual se podrá acceder a la interacción con el agente y la arquitectura interna del mismo.

El paquete agente: Contiene en su interior la arquitectura interna del agente. Y ofrece además el gestor del ciclo de vida del agente.

El paquete estado: Gestiona el estado actual del agente durante su ciclo de vida. Ofrece un gestor de estado.

El paquete itinerario: Registra los sitios a donde el agente va a migrar, contiene en su interior un gestor del itinerario y un objeto para el control del mismo.

El paquete identificador: Contiene en su interior el manejador para la identificación del agente en la plataforma HERMES.

El paquete conducta: Agrupa la conducta y las tareas que el agente ejecutaran a lo largo de su ciclo de vida.

ASH (Ambiente de Servicios HERMES)

El ASH, esta organizado lógicamente en paquetes, la relación que se existe entre cada componente se hace a través del paso de información entre cada elemento y paquete involucrado en un propósito específico. Su organización se hizo de forma jerárquica buscando un orden en roles y responsabilidades de cada componente.

La figura 15 muestra la distribución de los distintos paquetes que conforman el ASH.

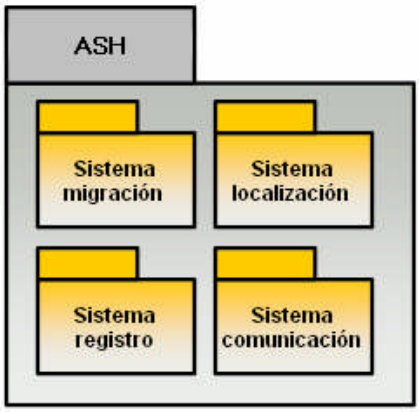

Ilustración 15. Composición de paquetes del nivel de ASH

El paquete ASH: Es la raíz del nivel 2 de la arquitectura del sistema HERMES, agrupa los subsistemas que ofrecen los diferentes servicios de la plataforma HERMES.

El paquete sistema migración: Este paquete agrupa los servicios de envió y recepción de los agentes, se ocupara del proceso de marshalling y unmarshalling y la respectiva serialización y deserialización de los objetos. Esta organizada por responsabilidades cada paquete tiene un propósito especifico.

El paquete sistema localización: Este paquete agrupa componentes que colaboran en el servicio de búsqueda de elementos como agentes y entornos. Para esto mantiene un gestor de localización y un componente para el registro.

El paquete sistema registro: Agrupa un solo componente que se encargara de la gestión del registro de cada elemento creado o adicionado a la plataforma HERMES.

El paquete sistema comunicación: Agrupa una serie de componentes que ofrecen el servicio de mensajeria, existe un gestor de comunicación y un elemento encargado de representar los mensajes que serán utilizados en el proceso de comunicación entre agentes, SMA y clones.

El ASH es el núcleo de la plataforma HERMES, agrupa los servicios de migración, comunicación, registro y localización. La composición interna y la interacción de cada componente del ASH se muestra en la figura 16.

El protocolo (TCP/IP) de comunicación usado para migrar agentes está integrado en el ASH, su implementación y uso es transparente al desarrollador de aplicaciones. Para la transmisión de agentes se utilizó sockets, la razón de su escogencia se debe a la facilidad que prestan en la implementación del empaquetamiento de información en agentes móviles y la garantía que la información llegue a su destino en el mismo orden en que ha sido transmitida.

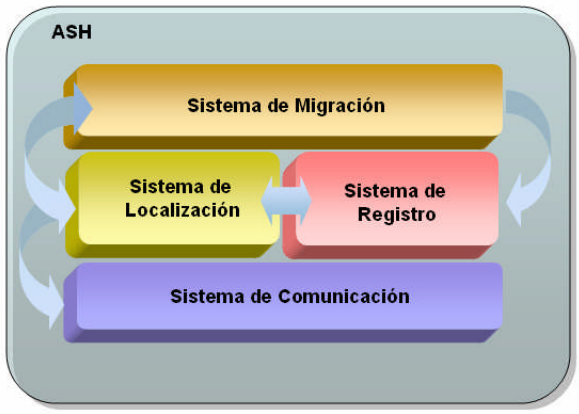

Ilustración 16. Arquitectura Interna del ASH

Existe una estrecha relación de dependencia del entorno con el ASH. El entorno solicita constantemente al ASH servicios de búsqueda, localización, registro, migración y comunicación para los agentes que residen en un entorno. Existe un

único componente ASH para todos los entornos creados en una aplicación multiAgente.

A continuación se detalla cada uno de los servicios que integran el ASH:

Sistema de Migración.

El sistema de Migración se ocupa del proceso de envió y recepción de los agentes. El proceso de migración ejecuta procedimientos como marshalling[17] y unmarshalling[18] de la información transferida por los agentes. Entre las responsabilidades más importantes que tiene el sistema de migración está la realización del proceso de serialización y deserialización.

El proceso de migración consiste en delegar al agente la responsabilidad de realizar un viaje a un destino determinado. Para llevar a cabo la realización de esta tarea tendrá que empaquetar la información necesaria para viajar a otras plataformas y entornos. Esta información debe contener los suficientes datos para restaurar e iniciar la ejecución de una tarea en el sitio escogido para viajar.

La composición interna del sistema de migración está distribuida en cuatro componentes vistos en la figura 17 (Serialización, Cliente, Servidor y Objetos Serializables). La cooperación e interacción de cada componente del sistema de migración permite que la información manipulada sea tratada adecuadamente, para ser enviada y recibida de forma apropiada. Obteniendo los datos necesarios en el momento requerido por el sistema MultiAgente.

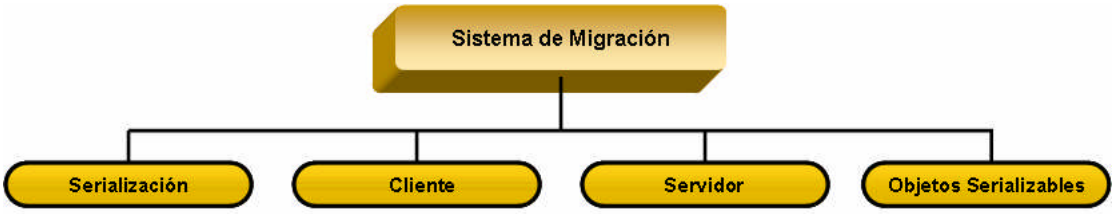

Ilustración 17. Composición del Sistema de Migración

La composición interna del sistema de migración está distribuida en cuatro componentes.

[17] Marshalling: Convertir una cadena de bytes a un formato entendible por la maquina para el proceso de envio de información por la red.

[18] Unmarshalling: Leer y Armar en una cadena de bytes el paquete de información recibido en el formato convertido por el proceso de marshalling.

Serialización:

Este componente se encarga de los procesos de escritura y lectura de la información de los agentes que se envían a un destino específico. Además contribuye con el cliente y el servidor en los procesos de marshalling y unmarshalling del agente. Además facilita los procesos de envió y recepción de mensajes remotos.

Cliente:

Este componente se ocupa de enviar el agente al destino escogido registrado en el itinerario. El proceso de marshalling es responsabilidad del Cliente y utiliza el mismo puerto para el envío de los agentes al servidor. Existe un cliente por cada agente viajero y solo será accedido en el momento de enviar el agente.

Servidor:

Este componente es el encargado de recibir a los agentes viajeros y reconstruir la información que traen. La información que se reconstruye es necesaria para reanudar la ejecución del agente en la máquina escogida. El servidor se ocupa del proceso de unmarshalling y se mantiene activo durante toda la aplicación esperando la llegada de un nuevo agente

Objetos Serializables:

Componente encargado de agrupar los objetos serializables propios de la plataforma HERMES. Es decir objetos que pertenezcan al API de JME Y JSE que no pueden ser serializables entre estas dos plataformas. Se implementó un objeto serializable “HashtableHERMES” el cual cuenta con las mismas características de una Hashtable, pero con el beneficio adicional de ser serializable.

Sistema de Registro

La función primaria del sistema de registro es inscribir[19] las identidades de entornos, agentes y clones de una aplicación multiagente. Otra función que tiene el sistema de registro es brindar seguridad a la información que se maneja en las aplicaciones multiagente.

El sistema de registro usa un directorio de registros con nombres e identificadores que le permiten distinguir, mantener y controlar las identidades de agentes, clones y entornos creados con la plataforma HERMES.

El mecanismo de seguridad que ofrece el sistema de registro permite mantener a salvo la integridad de datos de cualquier elemento[20] del sistema multiagente.

La seguridad que implementa el sistema de registro funciona como un filtro de identidades de cada elemento creado en una aplicación multiagente. El propósito de la seguridad es evitar la ambigüedad de la información y el colapso del sistema multiagente en el momento de inscribir agentes, clones y entornos.

El sistema de registro reconoce al nombre lógico de un elemento en el SMA como la única identidad válida para oficios de registro en el SMA. Es responsabilidad del desarrollador fijar el nombre lógico de un elemento del SMA. La gestión del nombre lógico le corresponde al sistema de registro.

Cuando es creado un agente, clon y entorno el sistema de registro valida el nombre lógico de dicho elemento con el resto de identidades inscritas en el registro del sistema. Si existen nombres lógicos iguales el sistema de registro impide la inscripción de ese elemento en el sistema e informa sobre el evento ocurrido; de lo contrario procederá a registrar el nuevo elemento.

Sistema de Localización

Componente que facilita la localización y búsqueda de agentes y clones en la aplicación multiagente en la plataforma HERMES. Permite acceder a información primaria de entornos, agentes y clones facilitando la interacción entre agentes y mejorando el desempeño del comportamiento de los mismos.

[19] Inscribir: Validar y Adicionar nombres lógicos en el sistema de registro del SMA
[20] Elemento: Actores o componentes del SMA (agente, clon y entorno).

La búsqueda y ubicación de elementos del SMA como agentes y clones se realiza a través del proceso de localización. Este proceso de localización usa un directorio de búsqueda que registra la ubicación donde residen los Agentes y clones.

El sistema de localización utiliza información que obtiene del sistema de registro, con el fin de construir su directorio de búsqueda de agentes y clones que facilite la ubicación de estos elementos en procesos de registro, comunicación y migración. Este directorio cumple además con el fin de mantener un sistema de nombrado de los agentes, donde se sabe a que entorno pertenecen y el identificador correspondiente a cada agente, así como los clones creados por cada agente padre y el entorno al que pertenecen.

La información necesaria para construir el directorio de búsqueda está constituida por el nombre lógico y su ubicación en el SMA (entorno). Los procesos de búsqueda y ubicación de agentes y clones en una aplicación multiagente son transparentes para el desarrollador.

Sistema de Comunicación

El componente encargado de la comunicación en HERMES es el sistema de Comunicación, su propósito es facilitar el proceso de interacción de los Agentes[21].

El sistema de comunicación proporciona un mecanismo de mensajería entre Agentes. Este mecanismo utiliza mensajes como elemento único para el envío de información entre agentes emisores y receptores. Crea un canal de comunicación para transmitir los mensajes de agentes y clones de modo que asigna un solo canal de comunicación por cada mensaje enviado; con el fin de impedir que mensajes no esperados se filtren en la interacción de agentes y clones. Este mecanismo de seguridad es transparente para el desarrollador de aplicaciones.

El Proxy del agente contribuye como intermediario para el envío y recepción de los mensajes. Colabora con la integridad en la transmisión de la información, de modo que los datos no sean modificados cuando se realiza el proceso de paso de mensajes entre agentes y clones.
Durante el proceso de interacción entre agentes el sistema de comunicación hace uso del nombre lógico de cada agente para distinguir a quien o quienes es enviado un determinado mensaje.

El Sistema de Comunicación requiere del sistema de localización para encontrar los agentes emisores y receptores de los mensajes enviados.

[21] Interacción de Agentes: Proceso de paso de mensajes entre agentes emisores y receptores.

En la plataforma HERMES se ha implementado dos modelos de comunicación necesarios para la interacción entre agentes y clones. La descripción de cada modelo se detalla a continuación:

- *Unicast*: El modelo de comunicación visto en la figura 18, muestra un agente emisor que envía un mensaje a un agente receptor del SMA. Si el receptor lo requiere responderá al agente emisor de la misma forma como recibió el mensaje.

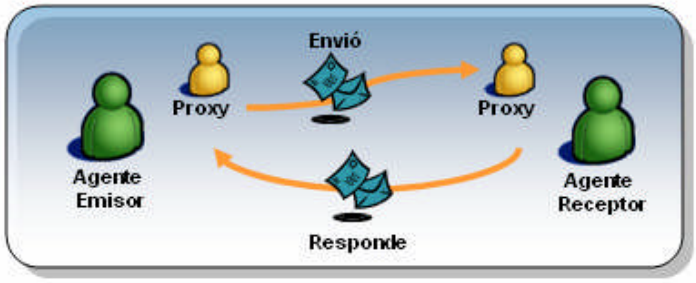

Ilustración 18. Modelo de Comunicación Unicast.

- *Broadcast*: El modelo de comunicación visto en la figura 19, muestra un agente emisor que envía un mensaje a un SMA receptor. Cada agente del SMA receptor puede responder el mensaje al agente emisor.

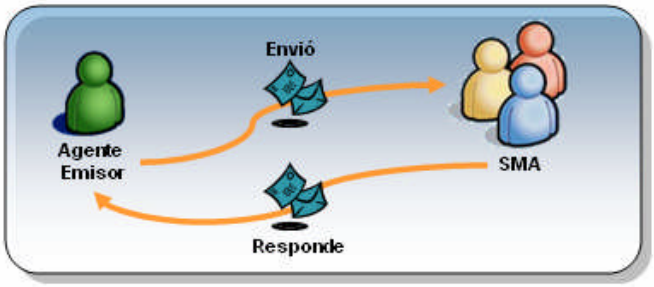

Ilustración 19. Modelo de Comunicación Broadcast.

Existe una variante de los modelos planteados anteriormente. Esta variante se presenta cuando ocurre un envío de un mensaje a un solo agente emisor y este puede replicar dicho mensaje a otro agente emisor o a un SMA.

La comunicación de agentes y clones se puede realizar entre máquinas remotas. Es necesario conocer la ubicación del agente receptor en la máquina remota para enviar el mensaje a través de la red al destino del agente receptor.

3.4. Ciclo de Vida de los Agentes HERMES

El esquema del ciclo de vida visto en la figura 20 describe los eventos[22] por los cuales pasa un Agente a lo largo de su ejecución en una aplicación multiagente. El ciclo de vida del agente comprende los pasos desde su creación, los estados de ejecución, los procesos de clonación y migración hasta llegar a su destrucción.

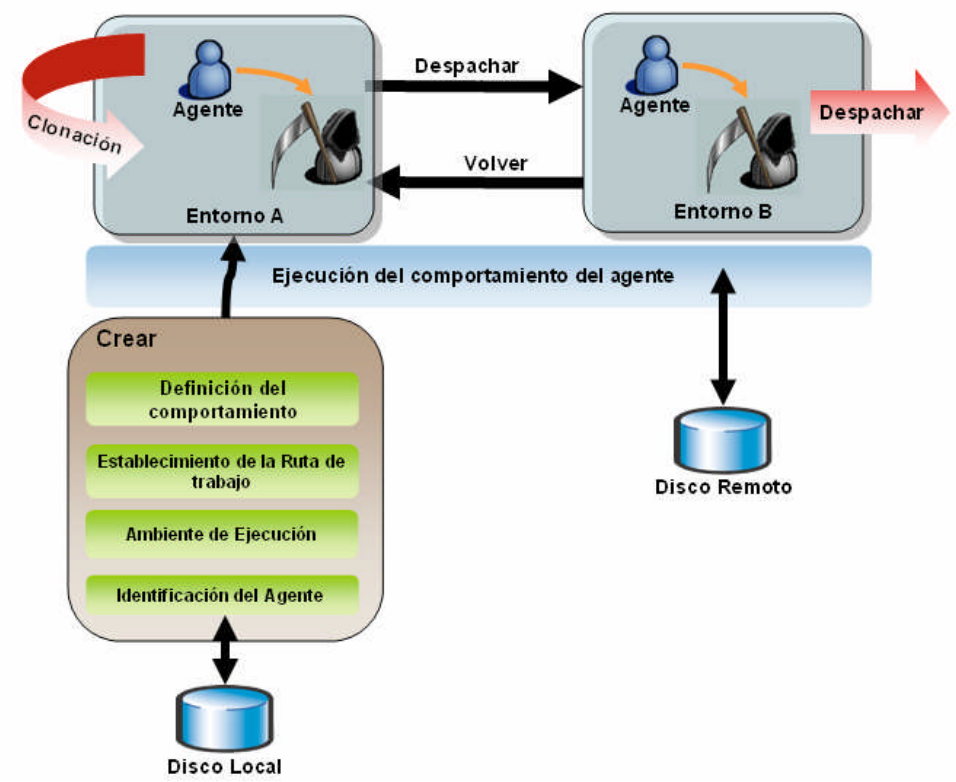

Ilustración 20. Esquema del Ciclo de Vida del Agente.

[22] Eventos: Acción o acto que sucede en el ciclo de vida del agente en una aplicación multiagente.

El ciclo de vida que se implementó en la plataforma HERMES comprende los siguientes eventos:

1. *creación*:

    a. *Identificar al agente:* El agente necesita tener una identificación. El nombre lógico debe ser fijado por el desarrollador en el momento de la creación del agente. El estado inicial se fija por defecto antes de inicializar la identificación del agente.

    b. *Crear el ambiente de Ejecución*: El agente requiere estar en un sitio específico para comenzar con su ejecución. El desarrollador adiciona el agente a un entorno escogido y creado previamente.

    c. *Establecer la Ruta de Trabajo*: El itinerario debe ser inicializado con los destinos donde se presume el agente viajará. Cada ruta o destino del agente debe ser fijada por el desarrollador.

    d. *Definición del Comportamiento*: Es necesario que se definan los comportamientos que el agente adoptará en su ciclo de vida. Cada comportamiento debe estar acompañado por la implementación de sus tareas.

2. *Ejecución del Agente:* Cada tarea del agente es ejecutada. Los atributos básicos como itinerario y estado del agente se ven modificados por cambios en el ambiente de ejecución. Durante la ejecución del agente se realizan los procesos de migración y comunicación.

3. *La clonación*: Evento opcional en el ciclo de vida del agente, donde se realiza una copia del Agente original. La creación de un clon busca tener un agente con las mismas características del original que cumpla las mismas actividades para las cuales fue creado el Agente original pero con una identificación diferente. Este evento es alternativo puesto que el programador podrá decidir si necesita o no Agentes clones.

4. *Migración:* Evento que comprende tres procesos secundarios*:*

    a. Despacho: Envío del agente a la máquina escogida para realizar el viaje. El itinerario brinda la dirección del destino o máquina residente.

b. Recepción: Arribo del agente a la máquina remota. Se ejecutan los comportamientos remotos del agente o se continúa con los que han sido interrumpidos.

c. Retorno: El Agente viajero regresa con la tarea cumplida y los resultados de la ejecución de la misma, para terminar un procedimiento si así lo requiere la lógica de la aplicación.

Cabe anotar que no siempre el agente debe regresar de su viaje. Si el comportamiento solo requiere un viaje sin retorno será suficiente para cumplir el objetivo y culminar así el ciclo de vida.

5. *Eliminación:* Característica inherente durante el ciclo de vida del agente, las tareas son finalizadas y los atributos básicos del agente son fijados en un valor nulo. Este evento sucede por varias razones:

   a. El comportamiento del Agente decide terminar con sus acciones.

   b. Un agente interrumpe la ejecución de otro agente terminando con su ciclo de vida.

   c. Por acción de un sistema que no esté basado en Agentes

   d. Por el sistema multiagente que involucra acciones como: fin del ciclo de vida del agente, falta de uso del mismo, violación de seguridad y shutdown (cierre de la plataforma HERMES).

3.5. Proceso de Migración

El proceso de migración visto en la figura 21 comprende los pasos[23] de envió, recepción, ejecución y retorno del Agente. A continuación se detallará cada uno de estos pasos y las actividades que debe realizar el agente para viajar y retornar a su sitio de partida.

[23] Paso: Acción o suceso ejecutado por el agente o SMA que ocurre en el proceso de migración.

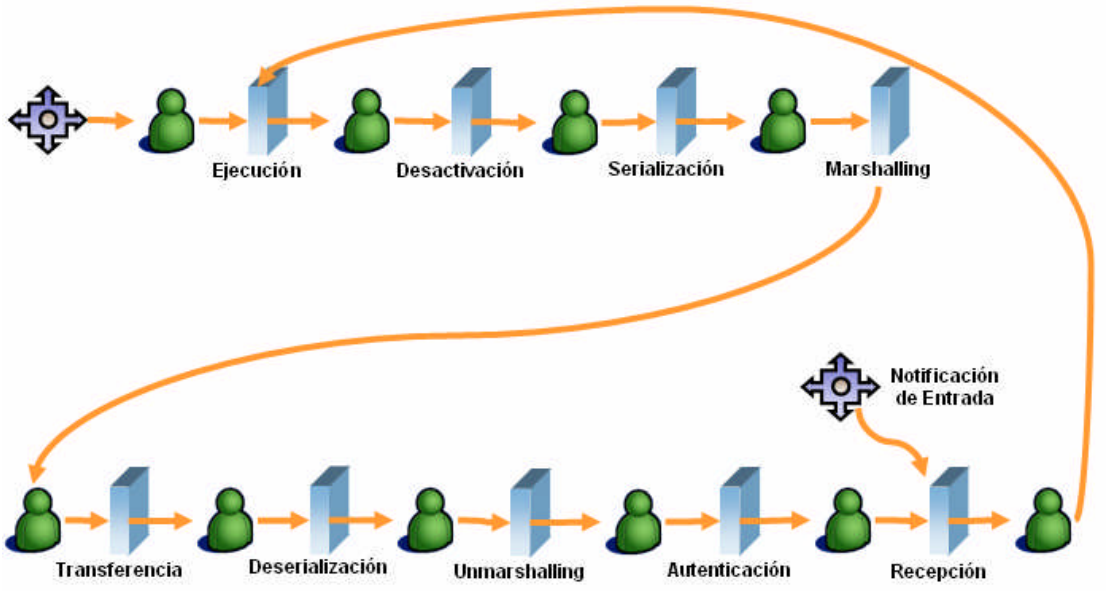

Ilustración 21. Proceso de Migración

El primer paso en el proceso de migración ocurre durante la ejecución del agente, al recibir la instrucción para realizar un viaje a un destino específico. De este modo el agente se prepara para empaquetar su información.

El siguiente paso desactiva la ejecución del agente, donde su estado y la información que maneja son fijados para el viaje.

La serialización es el paso que convierte al agente en un flujo de bytes para el envío a un destino específico. El siguiente paso es el marshalling del flujo de datos donde se fija un formato que depende de la maquina (móvil o fija) donde reside el agente específico para la transmisión de la información.

En la transferencia se realiza el envió de la información de un sitio a otro. De este modo el agente viaja al destino escogido para continuar con su ejecución.

La deserialización continua con el proceso de la transmisión de la información, en este paso se efectúa la conversión del flujo de bytes para su posterior recuperación. El paso de Unmarshalling lee el flujo de bytes en el formato enviado para convertirlo en datos reconocibles por la máquina.

La autenticación es el paso donde el sistema de registro verifica la identidad del agente y procede a registrarlo en el sistema.

En la recepción se registra el agente mediante una solicitud de notificación al sistema de registro. El siguiente paso comprueba el nombre lógico y el sitio de partida del agente para verificar su identidad. La información recogida en el paso de recepción permite reconstruir el estado de ejecución[24] de las instrucciones y procedimientos del agente.

El retorno del agente es el último paso en el proceso de la migración. Consiste en retornar al sitio de origen del agente con la información recogida del resultado de la ejecución de sus instrucciones y procedimientos. Para retornar con la información al sitio de partida del agente se debe repetir los procesos de envío, pero ahora hacia su sitio de origen y de este modo cerrar el proceso de migración.

La figura 22 representa el modelo completo del proceso del ciclo de vida del agente y su proceso de migración. Este modelo muestra cada uno de los eventos mencionados en el ciclo de vida y proceso de migración.

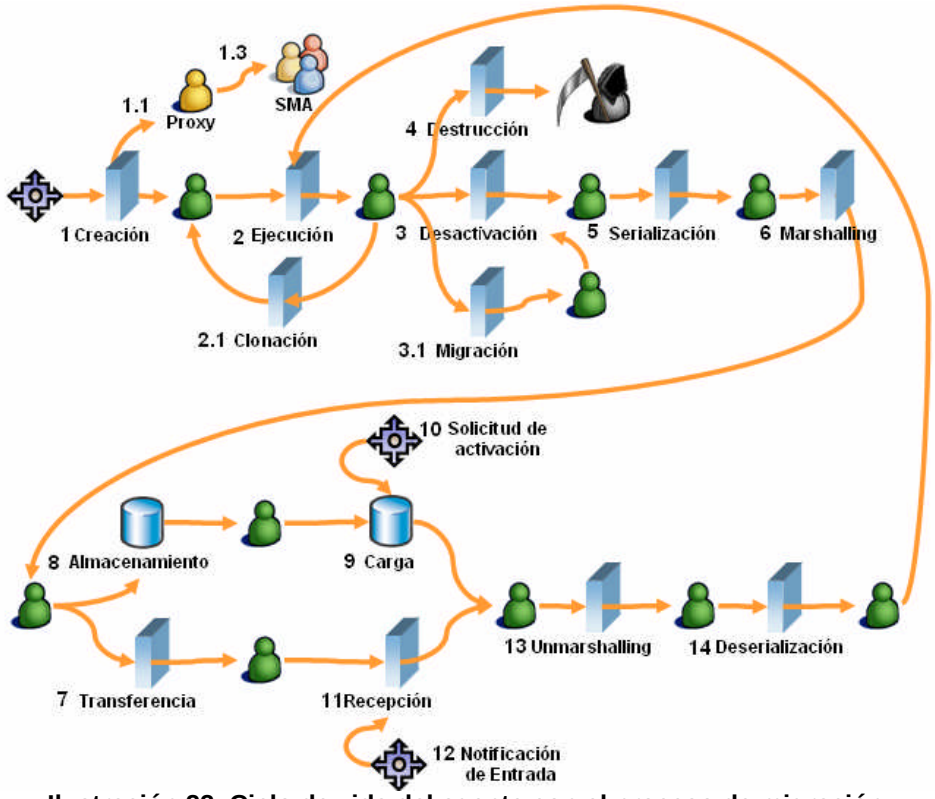

Ilustración 22. Ciclo de vida del agente con el proceso de migración

[24] Estado de Ejecución: Información necesaria para continuar con la ejecución del agente.

CAPITULO IV

AMBIENTE EXPERIMENTAL

4.1. Ambiente Experimental

El presente capítulo describe las pruebas hechas a la plataforma HERMES usando aplicaciones prototipo que validen la funcionalidad de los servicios que presta la API de HERMES. Se fijaron distintos parámetros de prueba como puntos de referencia para evaluar la funcionalidad del uso de la API en las aplicaciones prototipo. La aplicación prototipo descrita en este capítulo es la última realizada a la plataforma HERMES puesto que involucra todos los aspectos en cuanto a funcionalidad que ofrece HERMES, aspectos como la creación de entornos y agentes; comunicación y migración. El resto de aplicaciones desarrolladas se encuentran en el anexo B.

4.1.1. Descripción General del Sistema

El ambiente de experimentación de las pruebas realizadas a la plataforma HERMES cumple con las siguientes características:

1. Se cuenta con dos computadores de escritorio para implementar las aplicaciones prototipo y realizar las pruebas respectivas. Las características de los equipos son:

    - Computador de escritorio SUN Pentium 4 con un procesador de 2.4 Ghz, 512 mB de RAM y 80 Gb de disco Duro.

    - Computador de escritorio DELL Pentium 3 con un procesador de 863 Mhz, 128 mB de RAM y 20 Gb de disco duro

2. El sistema operativo seleccionado para los equipos de prueba es Windows XP.

3. El entorno de programación instalado en los equipos de prueba utilizado para desarrollar las aplicaciones prototipo es JBuilder 2005.

4. Cada equipo de prueba tiene instalado las dos plataformas de java JDK1.4 y WTK

5. Para simular los equipos móviles se utiliza el emulador que brinda JBuilder 2005 en sus paquetes de la plataforma J2ME 2.2.

6. La conexión entre los equipos de prueba es a través de una red LAN. En las instalaciones la universidad del cauca en la facultad de ingeniería electrónica.

7. Las versiones móvil y fija de la plataforma HERMES se encuentran en cada uno de los equipos de prueba.

8. Las aplicaciones prototipo están instaladas en cada equipo de prueba.

9. Los parámetros de prueba son aplicados a cada aplicación prototipo y su evaluación corresponde solamente a los resultados obtenidos en las pruebas hechas.

4.1.2. Restricciones y Recomendaciones:

Las aplicaciones para dispositivos móviles que usen la plataforma HERMES para implementar agentes móviles, se encuentran dentro de la gama media y alta por el tamaño del API de la plataforma HERMES, que esta alrededor de 93 kbytes.

En cuanto resolucion de pantalla y capacidad de navegación no hay problema. Solo existen restricciones por el limite de memoria que en algunos casos como la aplicación prototipo descrita en este capitulo requiere alta capacidad de memoria (alrededor de 2 mbytes de espacio).

Otro tipo de restricciones o recomendaciones optimas para implementar aplicaciones con HERMES son:

- Dispositivos con soporte a java
- Soporte a MIDP 1.0 y 2.0 y CLDC 1.1 (como requerimiento optino), depende del tipo de aplicación que se requiera.

4.2. Prototipo de Prueba Tienda de Compras

4.2.1. Descripción del Prototipo de Prueba

Este prototipo ha sido implementado con el fin de integrar en una sola aplicación la mayoría de los servicios que ofrece HERMES. La aplicación simula un sistema de compras asistido por dispositivos móviles. Existe una tienda que ofrece diferentes artículos a sus clientes, la tienda está ubicada en un dispositivo fijo que cuenta con una base de datos que guarda la información del inventario de la tienda. Cada vez que un cliente desea realizar alguna compra o simplemente consultar el inventario de la tienda, accede desde su dispositivo móvil a la base de datos de la tienda. La aplicación móvil permite ver en detalle cada artículo contenido en el inventario de la tienda y si el cliente lo desea, realizar la compra ofreciendo un precio alternativo al artículo deseado.

La aplicación fija que representa a la tienda, permite al administrador de la misma gestionar su base de datos. El administrador podrá ingresar nuevos artículos al inventario ya existente, actualizar los datos de los artículos y eliminar aquellos que ya hayan sido comprados. De tal modo que mantenga actualizado el inventario de los artículos ofrecidos a los clientes de los dispositivos móviles que accedan a los servicios de la tienda.

La aplicación tienda cuenta con un sistema multiagente que se encargará de realizar labores de actualización de interfaz de usuario, administración de la base de datos y negociación de artículos ofrecidos a los clientes.

La aplicación móvil de los clientes de la tienda, cuenta también con un sistema multiagente que realizará labores similares al SMA de la aplicación fija. El manejo de la interfaz del usuario en cuanto a listar los artículos disponibles por la tienda y las negociaciones con la misma son responsabilidad de los agentes implementados en la aplicación móvil.

Modelado del SMA de las aplicaciones Tienda y Cliente.

El sistema multiagente implementado en la aplicación fija de la tienda está constituido por tres agentes específicos:

- Agente Interfaz: Es un agente con características de agente controlador, por que tiene la responsabilidad de servir como puente entre los distintos agentes pertenecientes a la aplicación tienda. Además cumple con una labor de gestión e interacción con los agentes existentes en la aplicación tienda. El manejo de la interfaz de la aplicación es tarea de este agente, mantener la lista del inventario actualizada, mostrar el historial de ventas de la tienda y recoger la información del formulario de artículos en la interfaz de usuario, son sus tareas básicas como agente interfaz (ver figura 25). Es responsable también de la labor de interactuar o comunicarse con el agente Tienda para recibir las solicitudes del dispositivo móvil. Las responsabilidades del agente interfaz son vistas en la figura 23.

- Agente Tienda: Su responsabilidad primaria es enviar y recibir mensajes remotos a las aplicaciones móviles que accedan a la aplicación fija. Se comunica con el agente interfaz para obtener información sobre la lista de artículos disponibles en la tienda y poder así enviar un mensaje al móvil con el inventario actual de la tienda (ver figura 25). Las responsabilidades del agente interfaz son vistas en la figura 23.

- Agente Base de Datos: La tarea que tiene este agente es gestionar la base de datos de la aplicación tienda. El agente interfaz interactúa con el agente base de datos suministrandole la información del formulario de artículos para ser adicionados en la base de datos (ver figura 25). Otra tarea que tiene el agente base de datos es enviar la información necesaria de la gestión de la base de datos al agente interfaz para mantener actualizada la lista de inventario de la interfaz del usuario. Las responsabilidades del agente base de datos son vistas en la figura 23.

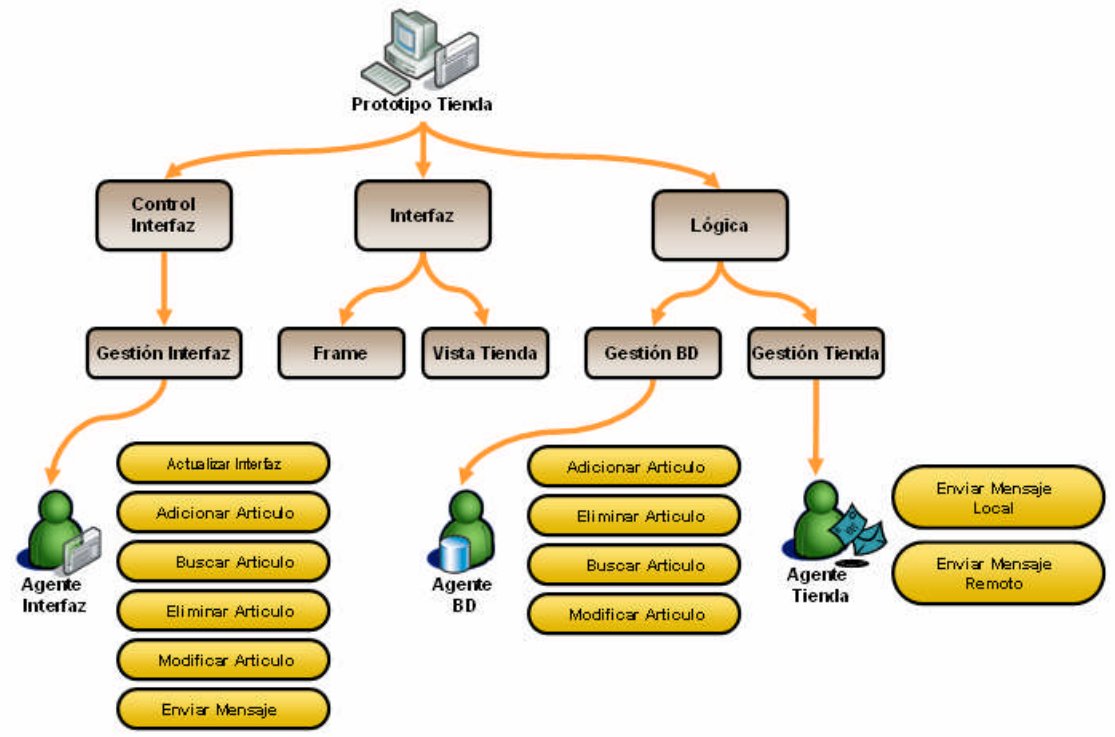

Ilustración 23. Modelo del SMA en la aplicación fija de Tienda

El sistema multiagente implementado en la aplicación móvil del cliente de tienda está constituido por tres agentes específicos:

- Agente Interfaz: Su papel es muy similar al agente interfaz de la aplicación fija, es un agente con características de agente controlador, por que tiene la responsabilidad de servir como puente entre los distintos agentes pertenecientes a la aplicación. Además cumple con una labor de gestión e interacción con los agentes existentes en la aplicación móvil. El manejo de la interfaz de la aplicación es tarea de este agente como mostrar la lista de artículos disponibles en la tienda e imprimir en la pantalla del móvil los detalles de las negociaciones hechas en la aplicación fija (ver figura 27). Las responsabilidades del agente interfaz son vistas en la figura 24.

- Agente Usuario: Su responsabilidad primaria es enviar y recibir mensajes remotos a la aplicación fija (ver figura 24), para obtener la lista del inventario de la tienda y pasarla al agente interfaz para que este la muestre en pantalla (ver figura 25).

- Agente Negociador: La tarea que tiene este agente es migrar a la aplicación fija y realizar la negociación del artículo escogido con el Agente Interfaz de la aplicación Tienda (ver figura 24). El agente negociador retorna al móvil con el resultado de la negociación y lo pasa al agente interfaz del móvil para que este le informe al usuario (ver figura 25).

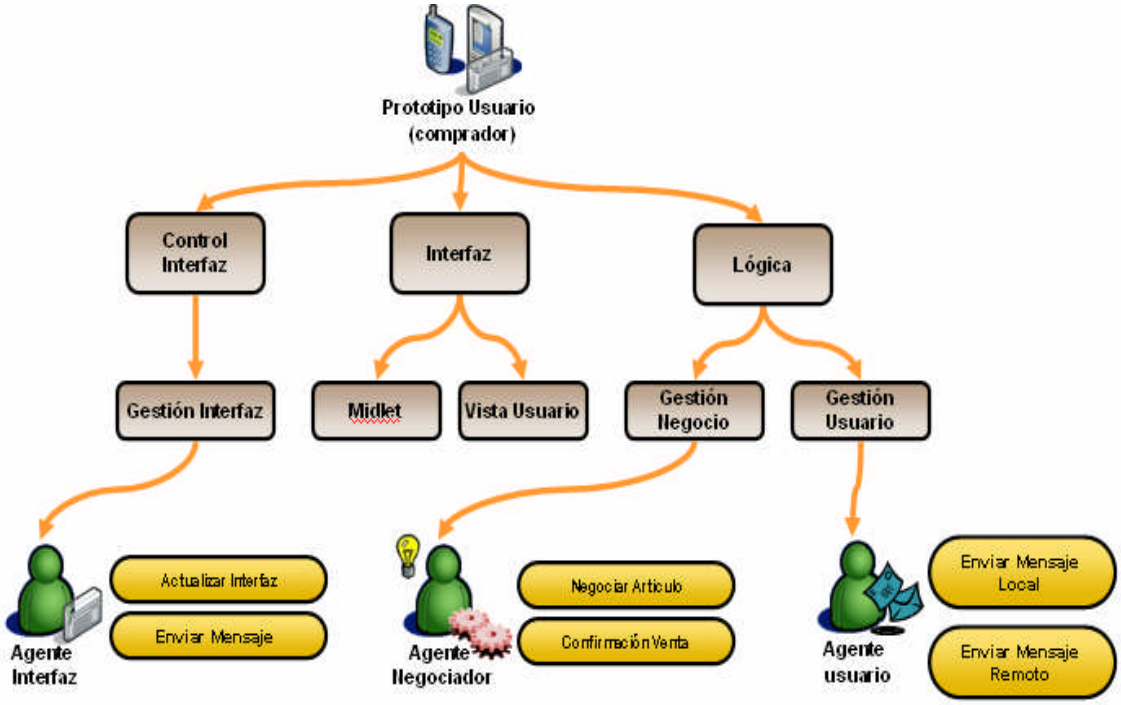

Ilustración 24. Modelo del SMA en la aplicación móvil de Cliente Tienda

Modelo de Interacción de los agentes del SMA de la aplicación Tienda y la aplicación Cliente Tienda.

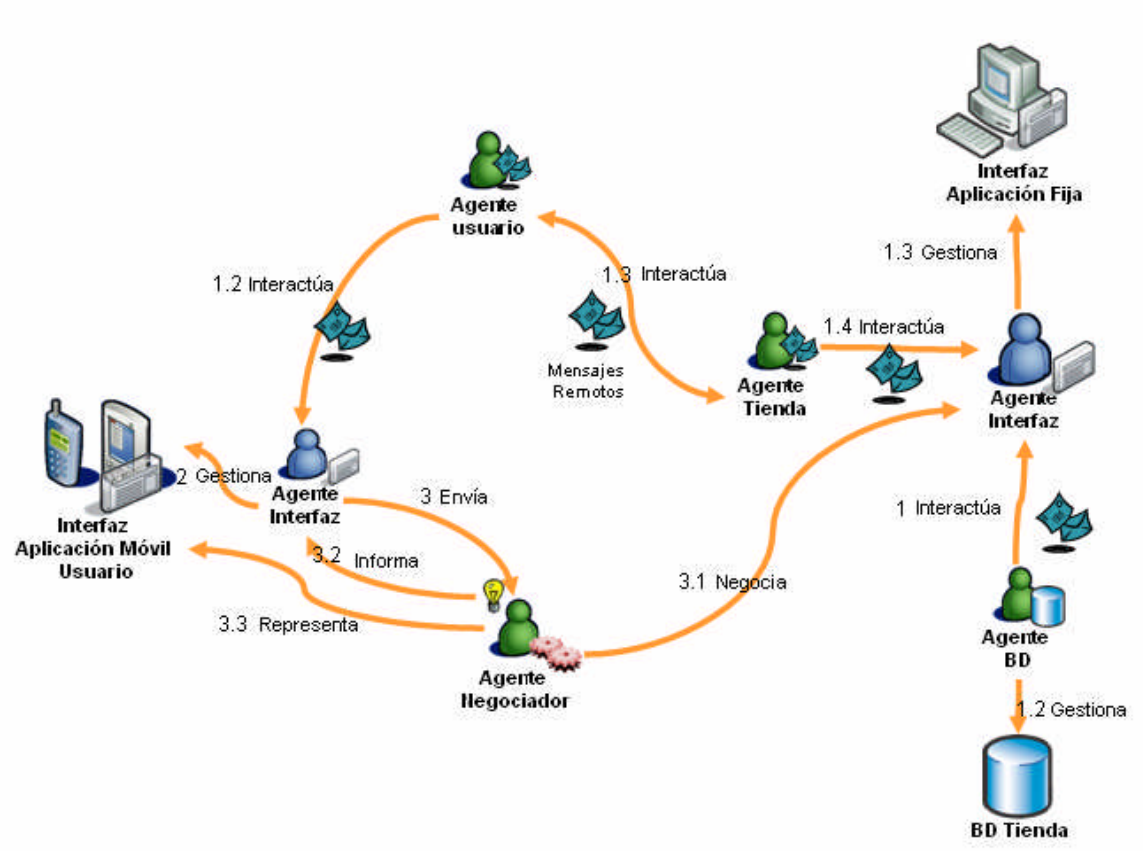

Ilustración 25. Modelo de Interacción del SMA de la aplicación Fija y la Móvil.

Objetivos de la Prueba:

- Comprobar en un caso real la capacidad que tiene la API de HERMES, de implementar aplicaciones funcionales tanto fijas como móviles.
- Verificar la capacidad de interacción que soporta la plataforma HERMES, entre distintos agentes con características diferentes.
- Comprobar la capacidad de cooperación y negociación que tienen los agentes de HERMES.
- Comprobar el correcto funcionamiento del proceso de migración, comunicación y gestión de tareas aplicado a los agentes implementados en las aplicaciones prototipo de Tienda y Cliente de Tienda.

Estrategia de evaluación al prototipo de prueba:

Para verificar que los objetivos planteados en el prototipo de Tienda y Cliente cumplan su propósito, se implementarán las siguientes funcionalidades a la aplicación:

- Enviar mensajes remotos de un dispositivo móvil a un fijo, con el fin de obtener el inventario de la tienda para conocer los artículos disponibles.
- Enviar Agentes móviles y comparar el precio de un artículo específico, con el precio que ofrece al cliente de la tienda.
- Otorgar la capacidad de negociación a los agentes de clientes para obtener el mejor precio de compra de un producto.
- la capacidad de negociación a los agentes de tienda para obtener el mejor precio de venta de un producto
- Comunicar los agentes remotos con agentes locales (interfaz)
- Mantener y gestionar un SMA en las aplicaciones fija y móvil.

Existe una única aplicación fija que representa a la tienda, que presta el servicio de ventas de productos a distintos clientes que acceden al inventario de los productos ofrecidos por la tienda a través de dispositivos móviles.

Las negociaciones de los productos se realizan en la aplicación fija, pero existe un agente por cada cliente que migra a la aplicación tienda desde un dispositivo móvil, de este modo si la negociación tuvo éxito y la compra se realizo el agente regresa a su sitio de partida (dispositivo móvil) con el resultado de la compra, una factura que muestra los detalles de la compra hecha. Si la negociación no tuvo éxito el agente regresa a su sitio de partida (dispositivo móvil) con los detalles de la negociación en un informe que presenta al cliente en la pantalla del móvil.

El prototipo de la aplicación fija de la tienda es una aplicación de escritorio con las siguientes características:

- Ofrece un menú de opciones en las plantillas de la interfaz de usuario de la aplicación tienda.

- Administrar Artículos:
  - La interfaz de usuario permite al administrador de la tienda ver la lista de artículos que dispone el inventario. (Figura 26, apartado 1).

- Adicionar, modificar y eliminar los datos de los artículos del inventario. Para esto cuenta con un formulario de detalles de los artículos. (ver figura 26, apartado 1).

- Para gestionar o administrar el inventario tiene tres botones que le permiten adicionar, modificar y eliminar los datos ingresados en el formulario. (ver figura 26, apartado 1).

- Artículos Disponibles:

  - La interfaz de usuario permite al administrador de la tienda ver la lista de artículos que dispone el inventario (Figura 38 apartado 2).

  - La lista permite seleccionar un artículo específico para ver los detalles del mismo (Figura 26 apartado 2).

- Historial de Ventas: La interfaz de usuario permite al administrador de la tienda ver los detalles de las operaciones de negociación de los distintos agentes de clientes que han hecho ofertas por algún artículo específico. Si la compra fue o no realizada, él informe sobre esta operación puede ser observado (Figura 26 apartado 3).

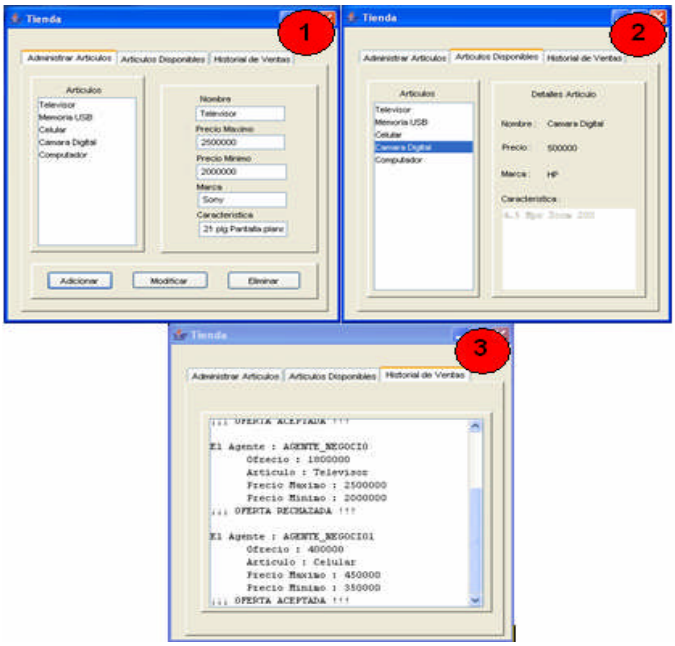

Ilustración 26. Interfaz de la Aplicación Fija de la Tienda.

El prototipo de la aplicación de cliente de la tienda es una aplicación de móvil con las siguientes características:

- Ofrece un menú de opciones en la pantalla principal del móvil. (Figura 39)

- Lista de Inventario:
  - Una vez es iniciada la aplicación móvil se carga la lista del inventario actualizado de la tienda (Figura 27 apartado 1).
  - El usuario selecciona un artículo específico para ver los detalles o realizar la compra (Figura 27 apartado 1).

- Detalles de artículo: En esta opción el usuario podrá conocer los detalles del artículo seleccionado como precio, características y el nombre (Figura 27 apartado 2).
- Realizar Compra: En esta opción el artículo seleccionado le es asignado un valor de compra con el cual el agente realizará la negociación. (Figura 27 apartado 3).

- Informe de Negociación: Los detalles de la negociación realizada en la aplicación fija de la tienda son vistos en la pantalla del móvil (Figura 27 apartado 1).

- Actualizar Artículos: En esta opción el Agente Interfaz actualiza la lista de inventario de la tienda mediante una solicitud de actualización a la aplicación fija, esta solicitud se delega al Agente Usuario.

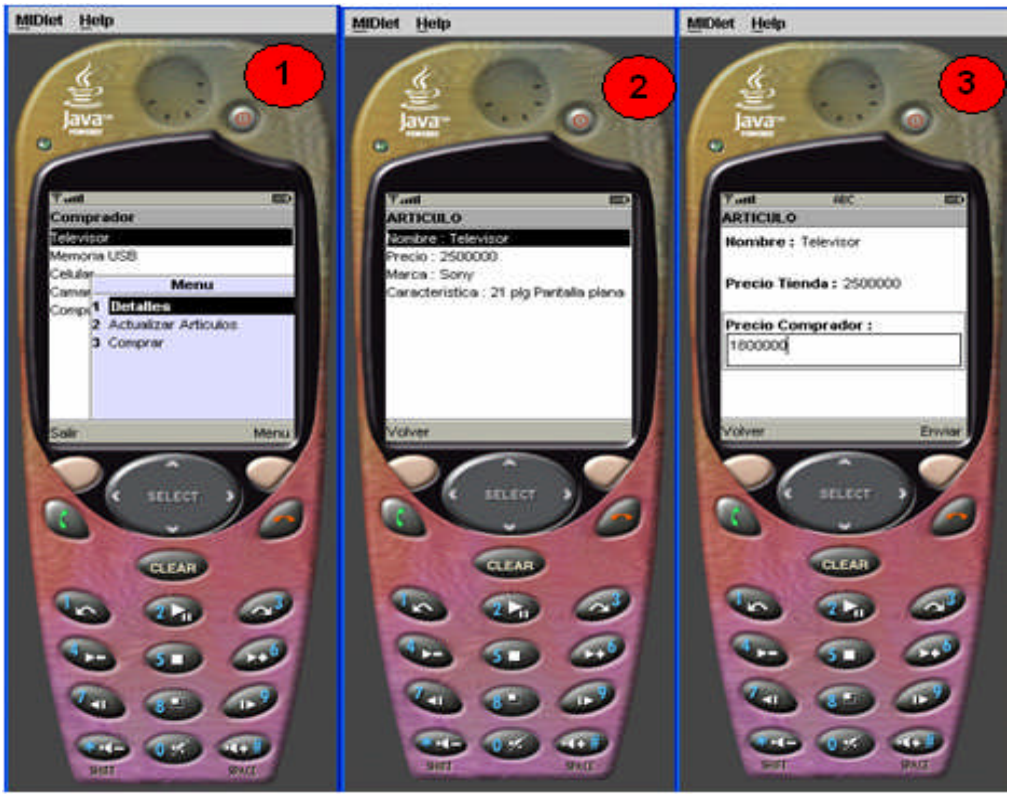

Ilustración 27. Interfaz de la Aplicación móvil del cliente Tienda.

4.5.2. Criterios de Experimentación

Los siguientes criterios de experimentación han sido tomados como puntos de evaluación para las pruebas realizadas a las aplicaciones de tienda y cliente de tienda.

- *Carga del cliente/servidor de la aplicación:* Se evalúa la confiabilidad al momento de inicializar el cliente/servidor de la aplicación tienda y cliente de tienda.
- *Validez del proceso de migración:* Se verifica la funcionalidad de los componentes de cliente y servidor para despachar y recibir agentes, así como la ejecución de los mismos en la aplicación fija y móvil.
- *Capacidad de migración de los agentes:* Se comprueba que los agentes seleccionados para viajar, efectivamente dejen la aplicación móvil y sean registrados en la aplicación fija y viceversa.
- *Capacidad de ejecución de los agentes viajeros:* Se verifica la capacidad de ejecución de una tarea remota del agente viajero cuando arriba al dispositivo fijo.
- *Envío y Recepción de Mensajes remotos:* Se comprueba que los mensajes remotos sean enviados y recibidos correctamente, verificando que la lista de inventario llegue correctamente a los dispositivos móviles que la solicitan.
- *Comprobación de Excepciones:* Se evalúa la capacidad de la plataforma para informar sobre alguna anomalía ocurrida durante la implementación y ejecución de las aplicaciones tienda y cliente tienda.

4.5.3. Detalles de Implementación

La aplicación fija tiene un SMA compuesto por tres agentes que se describen a continuación.

- La clase “agent_INTERFAZ” del agente interfaz se define de la siguiente forma:

```
public class agent_INTERFAZ extends agenteH {

// Instacia de los objetos que usara el agente interfaz
```

```
private actualizarInterfaz tareaActualizar;
   ……

// Constructora del Agente Interfaz
public agent_INTERFAZ(Frm_Tienda _interfaz) {

//  Se inicializa los atributos y objetos que se utilizaran   en
el  agente Interfaz
……
    }
public void manejadorMensaje(mensaje mensaje) {

// Confirmo que la firma del mensaje corresponde a una tarea
específica
if (mensaje.compararFirma("EnvioListaArticulos")) {
……..

// Ejecuto una tarea específica del comportamiento del  agente
this.ejecutarTareaEspecificaComportamientoLocal("GestionInterfaz",
"MensajeLocal");
        }
        ……
        if(mensaje.compararFirma("ActualizarLista")){
          …….
        }

        if (mensaje.compararFirma("AdicionarLista")) {
          ……
        }

        if (mensaje.compararFirma("EliminarArticuloLista")) {
          …….
        }

        if (mensaje.compararFirma("ModificarLista")) {
          ....……...
        }
    }
……
}
```

- La clase “agent_INTERFAZ” implementa las siguientes tareas:

- La tarea actualizar interfaz fija los datos recibidos desde la interfaz de usuario y mantiene actualizada la lista del inventario y el historial de ventas.

```
public class actualizarInterfaz extends tarea {
```

```
    private articulo myArticulo;
    private String myCodigoArticulo;
    ……..

    public actualizarInterfaz(Frm_Tienda _interfaz) {
    …….
        this.myArticulo = new articulo();
        this.myCodigoArticulo = "";
    }

    public void set_listaArticulos(HashtableHERMES _listaArticulos){
    ……..
     }

    public void set_Articulo(articulo _Articulo){
     ……..
    }

    public void set_CodigoArticulo(String _myCodigoArticulo){
     ……..
    }
    public void ejecutarTarea() {

        switch (this.opcion) {

        case 1:
            this.interfaz.cargarArticulos(this.listaArticulos);
            break;
        case 2:
            this.interfaz.cargarArticulo(this.myArticulo);
            break;
        case 3:
            this.interfaz.eliminarArticulo(this.myCodigoArticulo);
            break;
        }
        this.opcion = 0;
    }
……..
}
```

- La tarea AdicionarArticulo recoge los datos del formulario de interfaz y los pasa al agente de base de datos para que adicione un nuevo artículo en la base de datos.

```
public class ejecutarAdicionarArticulo extends tarea {

    private agent_BD myAgenteBD;

    public ejecutarAdicionarArticulo(agent_BD _myAgenteBD) {
```

```
        this.myAgenteBD = _myAgenteBD;
    }
    public void ejecutarTarea() {
this.myAgenteBD.ejecutarTareaEspecificaComportamientoLocal("GestionBase
Datos","Adicionar");
    }
…….
}
```

- La tarea BuscarArticulos, busca los Artículos en la base de datos, el agente interfaz ejecuta la tarea adecuada del agente Base de datos y este obtiene los artículos y los pasa al agente interfaz.

```
public class ejecutarbuscarArticulos extends tarea {

    private agent_BD myAgenteBD;

    public ejecutarbuscarArticulos(agent_BD _myAgenteBD) {
       this.myAgenteBD = _myAgenteBD;
    }

    public void ejecutarTarea() {

this.myAgenteBD.ejecutarTareaEspecificaComportamientoLocal("GestionBase
Datos","Buscar");
    }
```

- La tarea EliminarArticulos, el agente interfaz pasa un artículo específico al agente base de datos, para que este lo borre de la base de datos de la aplicación.

```
public class ejecutarEliminarArticulo extends tarea {

    private agent_BD myAgenteBD;

    public ejecutarEliminarArticulo(agent_BD _myAgenteBD) {
        this.myAgenteBD = _myAgenteBD;
    }
    public void ejecutarTarea() {

this.myAgenteBD.ejecutarTareaEspecificaComportamientoLocal("GestionBaseDa
tos","Eliminar");
    }
}
```

- La tarea ModificarArticulos, el agente interfaz pasa un artículo específico al agente base de datos, para que este lo actualice en la base de datos de la aplicación.

```
public class ejecutarModificarArticulo extends tarea {
    private agent_BD myAgenteBD;

    public ejecutarModificarArticulo(agent_BD _myAgenteBD) {
        this.myAgenteBD = _myAgenteBD;
    }
    public void ejecutarTarea() {
this.myAgenteBD.ejecutarTareaEspecificaComportamientoLocal("GestionBaseDa
tos","Modificar");

    }
}
```

- La tarea EnviarMensajeLocal, el agente interfaz envía mensajes locales a los agentes tienda y al agente negociador (cuando se encuentre en el dispositivo fijo).

```
public class enviarMensajeLocal extends tarea {

    public enviarMensajeLocal() {
    …….
    }
    public Object get_objetoMensaje(){
        return this.objetoMensaje;
    }
    public void set_objetoMensaje(Object _objetoMensaje){
        this.objetoMensaje = _objetoMensaje;
    }
    public String get_agenteDestino(){
        return this.agenteDestino;
    }
    public void set_agenteDestino(String _agenteDestino){
        this.agenteDestino = _agenteDestino;
    }
    public String get_firmaMensaje(){
        return this.firmaMensaje;
    }
    public void set_firmaMensaje(String _firmaMensaje){
        this.firmaMensaje = _firmaMensaje;
    }
    public String get_contenidoMensaje(){
        return this.contenidoMensaje;
    }
    public void set_contenidoMensaje(String _contenidoMensaje){
        this.contenidoMensaje = _contenidoMensaje;
    }
```

```
    public void ejecutarTarea() {
        this.My_Agente = (agent_INTERFAZ)this.obtenerAgente();
        this.MyMensaje = new
mensaje(firmaMensaje,contenidoMensaje,objetoMensaje);
        MyMensaje.set_origen(this.My_Agente.get_NombreLogico());
        MyMensaje.set_destino(this.agenteDestino);

        this.My_Agente.enviarMensaje(MyMensaje);
    }
```

- La clase “agent_BD”, es el agente base de datos que gestiona la información contenida en la base de datos de la aplicación fija y se define de la siguiente forma:

```
public class agent_BD extends agenteH {
    public agent_BD() {
    }

    public void adicionarComportamientosTareas() {
        adicionarComportamientoLocal("GestionBaseDatos");
        comportamiento myComportamiento
=obtenerComportamientoLocal("GestionBaseDatos");

    }

    public void set_ArticuloTareaAdicionar(String Nombre, String
PrecioMax, String PrecioMin, String Marca, String Caracteristicas){

        tareaAdicionar.set_Articulo(Nombre, PrecioMax, PrecioMin, Marca,
Caracteristicas);
    }

    public void set_ArticuloTareaModificar(int Codigo, String Nombre,
String PrecioMax, String PrecioMin, String Marca, String
Caracteristicas){

        tareaModificar.set_Articulo(Codigo, Nombre,PrecioMax,PrecioMin,
Marca, Caracteristicas);
    }


    public void set_CodigoArticuloTareaEliminar(int _Codigo){
        this.tareaEliminar.set_codigoArticulo(_Codigo);
    }
……….
}
```

- La clase “agent_BD” implementa las siguientes tareas:

- La tarea adicionarArticulo inserta un nuevo artículo en la base de datos de la aplicación.

```
public class adicionarArticulo extends tarea {

    ……….

    public adicionarArticulo() {
    ……..
    }

    public void set_Articulo(String _Nombre, String _PrecioMax,
String _PrecioMin, String _Marca, String _Caracteristicas){
      ……..
    }

    public articulo get_Articulo(){
       return this.myArticulo;
    }

    public void ejecutarTarea() {
    // Se implementa la sentencia de la base de datos para inserter
un Nuevo articulo
    ……..
   // Se implementa la conexión a la base de datos
        this.myBD.conexionBD_Postgres();

        if(this.myBD.DML(this.insert)){
  // Obtengo la llave primaria y confirmo la inserción del articulo
a la base de datos
    ……..

            }
        }
    }

    private void enviarArticulo() {
   ……..
  // Envío un mensaje al Agente Interfaz, con la información
suministrada por la base de datos
        this.myAgent.enviarMensaje(myMensaje);
    }
```

- La tarea eliminarArticulo elimina un artículo específico de la base de datos de la aplicación

```
public class eliminarArticulo extends tarea {
```

```
    public eliminarArticulo() {

    }

    public void set_codigoArticulo(int _codigoArticulo) {
        this.codigoArticulo = _codigoArticulo;
    }

    public void ejecutarTarea() {

    // Se implementa la sentencia de la base de datos para eliminar
un articulo
    ……..
   // Se implementa la conexión a la base de datos
        this.myBD.conexionBD_Postgres();

        if(this.myBD.DML(this.eliminar)){
            this.enviarCodigoArticulo();
        }
    }

    private void enviarCodigoArticulo() {
    // Envío un mensaje al Agente Interfaz, con la información
suministrada por la base de datos
        this.myAgent.enviarMensaje(myMensaje);
    }
……..
}
```

- La tarea buscarArticulo obtiene la lista de articulos de la base de datos.

```
public class buscarArticulos extends tarea {
……..

    public buscarArticulos() {
    ……..
    }
    public void ejecutarTarea() {

        this.buscar = "SELECT * FROM
\"Aplicacion_Tienda\".articulo;";
        this.myBD.conexionBD_Postgres;
        this.Rs = this.myBD.ObtenerDatos(this.buscar);
        this.cargarDatos();
        this.enviarlistaArticulos();
    }
    private void cargarDatos() {
```

```
    // Con los datos obtenidos de la búsqueda en la base de datos
los carga en una lista.
    }
    private void enviarlistaArticulos(){
    // Envío un mensaje al Agente Interfaz, con la información
suministrada por la base de datos
        this.myAgent.enviarMensaje(myMensaje);
    }
```

- La tarea ModificarArticulo cambia los datos de un artículo específico en la base de datos.

```
public class modificarArticulo extends tarea {
    public modificarArticulo() {
        ……
    }
    public void set_Articulo(int Codigo, String Nombre, String
PrecioMax, String PrecioMin, String Marca, String Carateristica){

// Fijo los datos del articulo que se va a modificar
    }

    public articulo get_Articulo(){
        return this.myArticulo;
    }
    public void ejecutarTarea() {
// Se implementa la conexión a la base de datos
this.myBD.conexionBD_Postgres();
            this.enviarArticulo();
        }
    }

private void enviarArticulo() {
// Envío un mensaje al Agente Interfaz, con la información
suministrada por la base de datos
this.myAgent.enviarMensaje(myMensaje);
    }
```

- La clase “agent_TIENDA”, es del agente Tienda que se encargará de enviar y recibir los mensajes remotos desde la aplicación fija a la móvil y viceversa.

```
    public agent_TIENDA() {
        …….
    }

    public void adicionarComportamientosTareas() {
   // Fijo los comportamientos y tareas que realizara el agente
        ……
```

```
    }
    public itinerario get_itinearioMensajeRetomo() {
        return this.itinearioMensajeRetomo;
    }

    public void set_itinearioMensajeRetomo(String NombreLogico,
String IP_Origen,
                                String IP_Destino) {
        this.itinearioMensajeRetomo.adicionarURL(NombreLogico,
IP_Origen, IP_Destino);
    }


    public enviarMensajeRemoto get_tareaMensajeRemoto(){
        return this.tareaMensajeRemoto;
    }

    public void set_tareaMensajeRemoto(String _firmaMensaje, String
_contenidoMensaje, String _agenteDestino, String _nombreLogicoURL,
HashtableHERMES _listaArticulos){

        // Fijo los atributos básicos del mensaje remoto
       ……..
     }

     public enviarMensajeLocal get_tareaMensajeLocal(){
         return this.tareaMensajeLocal;
     }
     public void set_tareaMensajeLocal(String _firmaMensaje, String
_contenidoMensaje, String _agenteDestino, Hashtable
_ListaArticulos) {

        // Fijo los atributos básicos del mensaje Local
        …….……..
     }
}

public void manejadorMensaje(mensaje mensaje) {

        if (mensaje.compararFirma("EnvioListaArticulos")) {
        ……
        }

        if (mensaje.compararFirma("SolicitaListaArticulos")) {
        ……
        }
    }
```

- La clase “agent_TIENDA” implementa las siguientes tareas:
  - La tarea enviarMensajeLocal envía un mensaje a cualquier agente registrado en la aplicación fija de la tienda.

```
public class enviarMensajeLocal extends tarea {

    public enviarMensajeLocal() {
        ……
    }

    public Object get_objetoMensaje(){
        return this.objetoMensaje;
    }
    public void set_objetoMensaje(Object _objetoMensaje){
        this.objetoMensaje = _objetoMensaje;
    }

    public String get_agenteDestino(){
        return this.agenteDestino;
    }
    public void set_agenteDestino(String _agenteDestino){
        this.agenteDestino = _agenteDestino;
    }

    public String get_firmaMensaje(){
        return this.firmaMensaje;
    }
    public void set_firmaMensaje(String _firmaMensaje){
        this.firmaMensaje = _firmaMensaje;
    }

    public String get_contenidoMensaje(){
        return this.contenidoMensaje;
    }
    public void set_contenidoMensaje(String _contenidoMensaje){
        this.contenidoMensaje = _contenidoMensaje;
    }

    public void ejecutarTarea() {
    // Fijo los agentes receptores del mensaje y el contenido del
mismo
      ……
      this.My_Agente.enviarMensaje(MyMensaje);
    }
```

  - La tarea enviarMensajeRemoto envía un mensaje al dispositivo móvil remoto para enviar la lista de inventario de la tienda.

```
public class enviarMensajeRemoto extends tarea {
    public enviarMensajeRemoto() {
    }

    public void set_ListaArticulos(HashtableHERMES
_listaArticulos){
        this.listaArticulos = new HashtableHERMES();
        this.listaArticulos = _listaArticulos;
    }
    public String get_nombreLogicoURL() {
        return this.nombreLogicoURL;
    }
    public void set_nombreLogicoURL(String _nombreLogicoURL) {
        this.nombreLogicoURL = _nombreLogicoURL;
    }
    public String get_agenteDestino() {
        return this.agenteDestino;
    }
    public void set_agenteDestino(String _agenteDestino) {
        this.agenteDestino = _agenteDestino;
    }
    public String get_firmaMensaje() {
        return this.firmaMensaje;
    }
    public void set_firmaMensaje(String _firmaMensaje) {
        this.firmaMensaje = _firmaMensaje;
    }
    public String get_contenidoMensaje() {
        return this.contenidoMensaje;
    }
   public void set_contenidoMensaje(String _contenidoMensaje) {
        this.contenidoMensaje = _contenidoMensaje;
    }
    public itinerario get_MyItineario() {
        return this.MyItineario;
    }
    public void set_MyItineario(itinerario _MyItineario) {
        this.MyItineario = _MyItineario;
    }
    public void ejecutarTarea() {
    // Fijo los agentes receptores del mensaje y el contenido del
mismo
       ……
       this.My_Agente.enviarMensajeRemoto(MyMensaje,
this.nombreLogicoURL);
    }
```

La aplicación Móvil tiene un SMA compuesto por tres agentes que se describen a continuación.

- La clase “agent_INTERFAZ” del agente interfaz se define de la siguiente forma:

```
public class agent_INTERFAZ extends agenteH {
    public agent_INTERFAZ(D_Comprador _interfaz) {
         //  Se inicializa los atributos y objetos que se
utilizaran  en  el  agente Interfaz

    }
    public void adicionarComportamientosTareas() {
       ……..
    }
    public enviarMensajeLocal  get_tareaMensajeLocal() {
        return this.tareaMensajeLocal;
    }
    public void set_tareaMensajeLocal(String _firmaMensaje, String
_contenidoMensaje, String _agenteDestino, Object _ListaArticulos) {
     // Fijo los atributos básicos del mensaje local
     …….
    }
    public void set_tareaMensajeLocal(String _firmaMensaje, String
_contenidoMensaje, String _agenteDestino, String _Confirmacion) {
     // Fijo los atributos básicos del mensaje local
     …….
      }
    public void iniciarMyAgente() {
        this.set_reanudar(true);
this.ejecutarTareaEspecificaComportamientoLocal("GestionInterfaz",
"MensajeLocal");
    }
    public void manejadorMensaje(mensaje mensaje) {

        if(mensaje.compararFirma("ActualizarLista")){
            // Ejecuto una tarea especifica de acuerdo a la firma
            …….

this.ejecutarTareaEspecificaComportamientoLocal("GestionInterfaz","
Actualizar");
        }
        if (mensaje.compararFirma(Boolean.TRUE.toString())) {
            ……
        }
        if (mensaje.compararFirma(Boolean.FALSE.toString())) {
            …….
        }
    }
```

- La clase “agent_INTERFAZ” implementa las siguientes tareas:

  - La clase actualizarInterfaz es una tarea que muestra la lista del inventario disponible de la tienda en la pantalla del dispositivo móvil.

```
public class actualizarInterfaz extends tarea {
    …….

    public actualizarInterfaz(D_Comprador _interfaz) {
     ……
    }

    public void set_listaArticulos(HashtableHERMES
_listaArticulos){
     …….
    }


    public void set_Articulo(articulo _Articulo){
      ……
    }

    public void set_CodigoArticulo(String _myCodigoArticulo){
      …….
    }


    public void ejecutarTarea() {

            this.interfaz.cargarArticulos(this.listaArticulos);
    }
```

  - La clase enviarMensajeLocal es una tarea que envía mensajes locales a los agentes registrados en la aplicación móvil.

```
public class enviarMensajeLocal extends tarea {

…….

    public enviarMensajeLocal() {
    ……
    }

    public Object get_objetoMensaje(){
        return this.objetoMensaje;
    }
    public void set_objetoMensaje(Object _objetoMensaje){
```

```
        this.objetoMensaje = _objetoMensaje;
    }

    public String get_agenteDestino(){
        return this.agenteDestino;
    }
    public void set_agenteDestino(String _agenteDestino){
        this.agenteDestino = _agenteDestino;
    }

    public String get_firmaMensaje(){
        return this.firmaMensaje;
    }
    public void set_firmaMensaje(String _firmaMensaje){
        this.firmaMensaje = _firmaMensaje;
    }

    public String get_contenidoMensaje(){
        return this.contenidoMensaje;
    }
    public void set_contenidoMensaje(String _contenidoMensaje){
        this.contenidoMensaje = _contenidoMensaje;
    }

    public void ejecutarTarea() {

      // Fijo los atributos básicos del mensaje local
      …….
       this.My_Agente.enviarMensaje(MyMensaje);
    }
      ……
}
```

- La clase “agent_NEGOCIADOR” del agente negociador se define de la siguiente forma:

```
public class agent_NEGOCIO extends agenteH {
……..
    public agent_NEGOCIO() {
    …….
    }

    public void adicionarComportamientosTareas() {

    //   Defino los comportamientos y tareas que se necitaran en la
ejeucion del agente
          ……..
    }

    public boolean get_confirmacion() {
```

```
        return this.confirmacion;
    }

    public void set_tareaNegociar(String firmaMensaje, String
precioOfrecido, String agenteDestino, String codigoArticulo) {
        //  Fijo los atributos basicos de la negociación como
precio, nombre, codigo del articulos etc.
        …….
    }
```

```
 private void set_tareaConfirmacion(String precioOfrecido, String
agenteDestino, String codigoArticulo){
      // Defino los atributos de la confirmacion de la negociación
      ……
    }

    public void set_my_Itinerario(String _NombreLogico, String
_IP_Origen, String _IP_Destino, String _Entorno_Origen, String
_Entorno_Destino){
   …….
    }

    public void manejadorMensaje(mensaje mensaje) {

        if (mensaje.compararFirma("EnvioNegociacion")) {
        //  Si la firma del mensaje es para la tarea de negociación
se ejecuta la tarea
            ……..
            this.despachoRemoto("FIJO");
        }

         if (mensaje.compararFirma(Boolean.TRUE.toString())) {
            ……
            this.despachoRemoto("MOVIL");
        }


        if (mensaje.compararFirma(Boolean.FALSE.toString())) {
            ……
            this.despachoRemoto("MOVIL");
        }
    }

    public void readObject(ObjInputStream objInputStream) throws
IOException {
        this.confirmacion = objInputStream.readBoolean();
        this.leerAgente(objInputStream);
    }
```

```
    public void writeObject(ObjOutputStream objOutputStream) throws
IOException {
        objOutputStream.writeBoolean(this.confirmacion);
        this.escribirAgente(objOutputStream);
    }
}
```

- La clase “agent_NEGOCIADOR” implementa las siguientes tareas:
  - La clase negociarArticulo es una tarea que se ocupa de la negociación del precio de un artículo específico. El resultado de la negociación es enviado a través de un mensaje.

```
public class negociarArticulo extends tarea {
 ……….
    public negociarArticulo() {
    ……….
    }

// Implemento los metodos para fijar y retornar los atributos
basicos de la negociación
……….

    public void ejecutarTarea() {

       // Fijo los atributos basicos de la negociación para enviar
los resultados a traves de un mensaje
        …….
        this.My_Agente.enviarMensaje(this.MyMensaje);
    }

    public void readObject(ObjInputStream objInputStream) throws
IOException {
        …….
        this.leerTarea(objInputStream);
    }

    public void writeObject(ObjOutputStream objOutputStream) throws
IOException {
     ……..
        this.escribirTarea(objOutputStream);
    }
}
```

  - La clase confirmacionVenta es una tarea que se ocupa de informar sobre los detalles de la negociación de las ventas en la tienda.

```
public class confirmacionVenta extends tarea {
    public confirmacionVenta() {
```

```
     …….
    }

// Implemento los metodos para fijar y retornar los atributos
basicos de la confirmacion de la venta
……
    public void ejecutarTarea() {

    //  Valido el valor de la confirmacion de la venta y procedo a
   fijar los valores de los atributos de la              //
   negociación para ser enviados por un mensaje
      ……
             this.My_Agente.enviarMensaje(this.MyMensaje);
    }
    public void readObject(ObjInputStream objInputStream) throws
IOException {
          ……
        this.leerTarea(objInputStream);
    }
   public void writeObject(ObjOutputStream objOutputStream) throws
IOException {
    ……..
        this.escribirTarea(objOutputStream);
    }
}
```

El agente Usuario Tienda se implementa de la misma forma que el Agente Tienda de la aplicación fija pero ahora es para que cumpla el rol de agente mensajero (mensajes remotos y locales) para la aplicación móvil.

4.5.4. Resultados Obtenidos

- *Carga del cliente/servidor de la aplicación*:

  Se comprobó la capacidad de los componentes cliente y servidor en las aplicaciones tienda y cliente de tienda. El servidor estuvo siempre activo a la espera de la llegada de un mensaje remoto o de un agente viajero. Igualmente el cliente funcionó correctamente al enviar mensajes remotos y agentes viajeros. A pesar que existiesen varios móviles activos en el mismo momento no se presentó ningún conflicto con los componentes de servidor y cliente de cada aplicación móvil (ver figura 41).

- *Validez del proceso de migración y capacidad de Ejecución de los agentes Viajeros:*

Se realizaron continuas pruebas donde se enviaron varios mensajes remotos desde el dispositivo móvil al fijo obteniendo la lista del inventario de la tienda y siempre fue correcto el envió y recepción del mensaje remotos. Se verificó esto observando la información que muestra la pantalla de los dispositivos móviles y comparando su contenido con la lista que muestra la aplicación fija del inventario de la tienda. Esta prueba siempre resultó correcta.

El proceso de negociaciones de los agente móviles siempre arrojó resultados satisfactorios. Se comprobó que los agentes negociadores de las aplicaciones móviles viajaron con la información correcta suministrada por el usuario móvil y efectivamente realizaron la negociación en la aplicación fija con otros agentes (ver figura 40). La manera de comprobar este hecho era verificando el historial de ventas de la aplicación fija de la tienda y comparando con la información de los detalles de negociación mostrados en la pantalla de la aplicación móvil. El resultado de estas pruebas fue el esperado comprobando la validez del proceso de migración y de ejecución remota de los agentes en las aplicaciones tienda y cliente de tienda.

- *Envío y Recepción de Mensajes remotos* :

  Los mensajes remotos enviados desde las aplicaciones móviles y recibidos desde la aplicación fija, arrojaron resultados correctos, este hecho se comprobó verificando que la lista del inventario siempre llegaba actualizada a los móviles con la información correcta de cada artículo disponible en la tienda. Para verificar el contenido de los mensajes se comparan los contenidos de la lista de inventarios mostrada en las pantallas de los dispositivos móviles (ver figura 39 apartado 1), con la información que muestra la aplicación fija de la tienda (ver figura 38 apartado 1).

- *Comprobación de Excepciones:*

  Los mensajes en consola que ofrece la plataforma HERMES siempre informaron sobre anomalías y hechos sucedidos en la ejecución de las aplicaciones tienda y cliente de tienda, como la inicialización de los componentes cliente y servidor, arribo y despacho de agentes móviles y mensajes remotos, entre otros (ver figura 41).

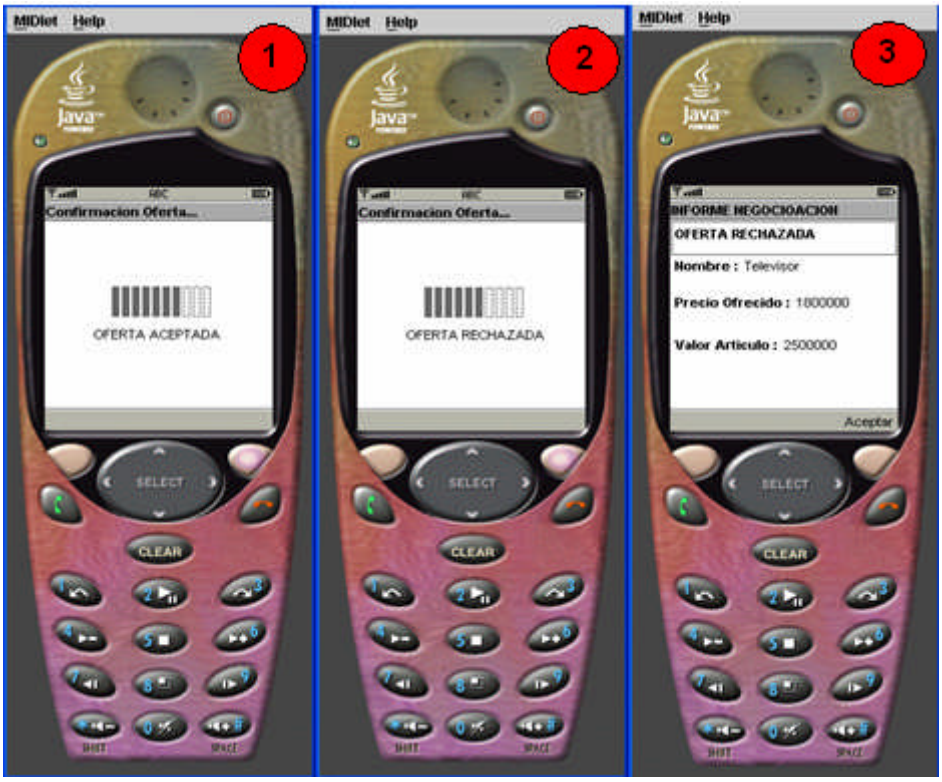

Ilustración 28. Resultados de la negociación en la aplicación fija.

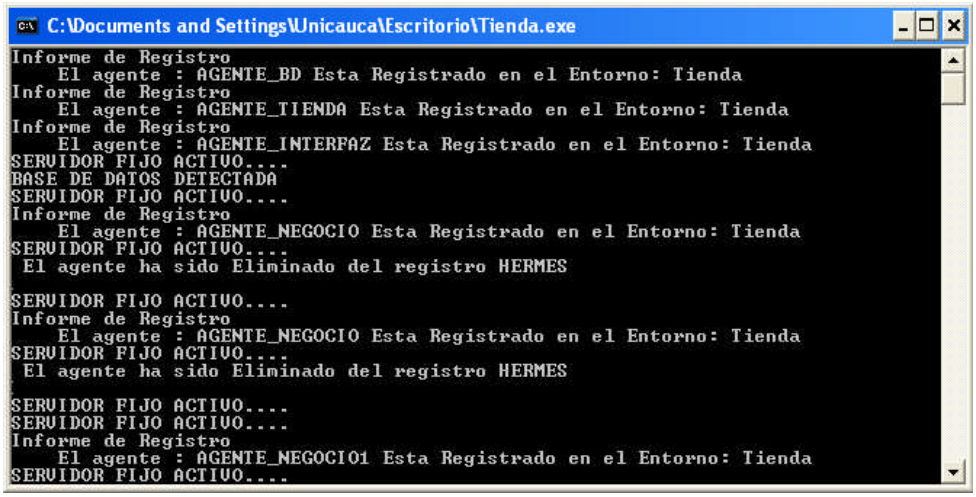

Ilustración 29. Excepciones de la plataforma HERMES
En la aplicación cuatro.

CAPITULO V

CONCLUSIONES Y PERSPECTIVAS

5.1. Conclusiones

El objetivo principal del proyecto HERMES fue desarrollar una plataforma que soportara la migración de agentes de dispositivos móviles a fijos y viceversa.

Se obtuvo un producto que ofrece una mayor flexibilidad en el momento de realizar actualizaciones o nuevas versiones a la API existente, porque cuenta con una arquitectura escalable que permite adicionar nuevos servicios y componentes, así como la mejora de los ya existentes, sin que afecte el desempeño de la plataforma.

Se obtuvo una API que brinda herramientas (clases y métodos) para implementar agentes y ambientes de ejecución propicios en la construcción de aplicaciones multiagente. Además se alcanzó el desarrollo de mecanismos de comunicación entre agentes y clones implementados con la API de HERMES.

El mayor avance que se obtuvo en la plataforma HERMES fue brindar mecanismos de migración de agentes entre dispositivos móviles y fijos. Obteniendo resultados satisfactorios en la transferencia de agentes.

Integrar diferentes conceptos de agentes móviles y plasmarlos en una herramienta brinda una sólida base para la construcción de nuevas y mejores versiones del prototipo alcanzado.

A medida que se avanzaba en el desarrollo del proyecto surgieron diversas dificultades que se fueron solventando satisfactoriamente. El problema más

importante fue la incompatibilidad de las API de JME y JSE. Problema que se solucionó implementando dos versiones de cada plataforma HERMES.

Otro problema encontrado a lo largo del desarrollo del proyecto, fue el inconveniente en la ejecución de procedimientos en máquinas distintas con API's incompatibles; problema que se solucionó con el manejo del concepto de tareas remotas y locales.

Se realizaron pruebas reales de la plataforma HERMES con pequeñas aplicaciones que buscaban en su diseño evaluar la funcionalidad de cada clase y método perteneciente a la API. Los resultados obtenidos demostraron que la funcionalidad provista a la plataforma HERMES cumple con las expectativas esperadas.

El desarrollo de un proyecto que involucre la construcción de una API para una plataforma de cualquier característica, es un proceso que debe ser estudiado con mucho detalle antes de comenzar con la labor de desarrollo e investigación. Se debe tener un estudio concienzudo sobre las plataformas existentes, conocer las bondades y falencias de proyectos existentes para retroalimentar el proyecto y alcanzar el desarrollo de una herramienta software de alta calidad y desempeño.

Encontrar un lenguaje de programación apropiado que facilite la implementación del proyecto y tener los conceptos muy claros acerca de las características funcionales que se desean aportar a la plataforma en cuestión.

Se obtuvo un producto que cumple con los requerimientos mínimos de un prototipo de una plataforma de agentes. Que según el estandar FIPA reconoce una plataforma de agentes cuando permite [FIPA 2002]:

- *Desarrollar agentes*: Es necesario que la plataforma tenga una API que ofrezca clases y metodos para la creación e inicializacion de los atributos basicos de un agente.

- *Ejecutar agentes moviles*: Es necesario contar con un ambiente de ejeución para que los agentes desarrollados puedan cumplir con su ciclo de vida.

Teniendo en cuenta lo anterior y tomando como base el concepto manejado por plataformas ampliamente reconocidas como Aglets, ARA, Telescript entre otras que cumplen también con las especificaciones de FIPA. Se puede concluir que HERMES es una plataforma de Agentes en su primera version.

5.2. Perspectivas

El desarrollo de investigaciones en el campo de los agentes móviles ofrece amplias alternativas para los desarrolladores e investigadores de este campo. La plataforma construida es aún una versión muy simple. Por lo tanto debe ser complementada y actualizada en su API y en otros factores de implementación.

Las nuevas versiones que se implementen en la plataforma HERMES deben estar encaminadas hacia las siguientes características:

- Implementar mecanismos de seguridad para el acceso a métodos y procedimientos propios de la plataforma HERMES, que deben ser ocultados al desarrollador para evitar conflictos con el desempeño de la plataforma.

- Construir módulos de seguridad para las transferencia de agentes a través de la red, de modo que sea inviolable la información que los agentes llevan en el trayecto de un sitio a otro.

- Aumentar el desempeño y funcionalidad de los agentes, profundizando en los conceptos teóricos de agentes móviles de tal modo que sean implementados nuevos componentes que ofrezcan mayores características como proactividad, reactividad, inteligencia etc. Además mejorar las existentes como la movilidad, planeación y ejecución de sus comportamientos y tareas.

- Adicionar patrones de diseño (experto, maestro/esclavo, itinerario etc.) en la implementación de agentes móviles, que permitan aumentar la funcionalidad de las aplicaciones construidas con la API de HERMES.

- Implementar un mecanismo que soporte la carga dinámica de clases.

- Mejorar el mecanismo de Serialización, buscando automatizar completamente los procesos de escritura y lectura de los datos de los agentes implementados con la API de HERMES.

- Construir una sola versión de la plataforma HERMES, donde las API de JSE y JME puedan ser compatibles en una sola plataforma.

BIBLIOGRAFIA Y GLOSARIO

[Bauer 2005] BAUER Bernhard y ODELL James. UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard. Journal of Engineering Applications of Artificial Intelligence Volume 18, Issue 2 , Marzo 2005, Pages 141-157.

[Barrera 2005] Barrera Fernando Campo y David Alejandro Martínez Vásquez, (2005) Aplicaciones Para Dispositivos Móviles Utilizando La Tecnología De Agentes, Universidad del Cauca (Colombia).

[Bigus 2002] BIGUS, J y BIGUS, J P. Constructing Intelligent Agents Using Java. Wiley Computer Publishing. 2002.

[Bond 1998] BOND, H. y GASSER, L. Readings in Distributed Artificial Intelligence. Morgan Kaufmann, San Mateo, CA, 1998.

[Brooks 1996] BROOKS, R.A. A robust layered control system for a mobile robot. IEEE journal of Robotics and Automation 2:14-23. 1996.

[Castañeda 2004] Josefina Castañeda Camacho y Domingo lara Rodríguez (2004) Performance of a New Microcell/Macrocell Cellular Architecture with CDMA Access. IEEE Veh. Technol. Conference. Pp 34-42

[Donald 2002] Donald C. Cox (2002) Wireles Personal Comunications: What Is It? IEEE Pers. comm. pp. 20-30.

[Douglas 2001] Douglas N. Knisely, Sarath Kumar, et. al. (2001) Evolution of Wireless Data Services: IS-95 to cdma2000. IEEE Comm. Magazine, pp.140-149.

[Diez 2002] Diez Andino Sancho (2002) Diseño e Implementación de un mecanismo de un servidor HTTP y mecanismo de serialización en Dispositivos Móviles, Universidad Carlos III de Madrid (España).

[Ferguson 1992] FERGUSON, I. A., "Towards an architecture for adaptive, rational, mobile agents". In Werner, E. and Demazeau, Y., editors, Decentralized AI 3. Proceedings of the Third European Workshop on Modelling Autonomous Agents and MultiAgent Worlds (MAAMAW91), pp. 249-262. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992

[Garcia 2002] Garcia Carlos Rubio, Lopez Andres Marin, Campo Celeste Maria, Plataforma de Agentes en Terminales de Telefonía Móvil. Departamento de Ingeniería Telemática Carlos III de Madrid (España). 2002

[GRAY 1997] Gray Robert S. Agent Tcl: a flexible and secure mobile agent system. Tesis Doctoral, Darmouth college, Hanover 1997.

[FIPA 2002] FIPA. Abstract Architecture Specification SC00001L. Foundation for Intelligent Physical Agents, Geneva, Switzerland, diciembre, 2002.

[Franklin 1996] FRANKLIN, S. and GRAESSER, A. Is it an agent, or just a program A taxonomy for autonomous agents. In Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Budapest, Hungary, Aug. 1996. ECAI'96, Springer-Verlag: Heidelberg, Germany. URL: http://www.msci.memphis.edu/franklin/AgentProg.html.

[Habitat 2005] Habitat-ProTM Environment, Agents Inspired Technologies S.A, University of Girona, Girona, Spain, 2005, http://www.agentsinspired.com

[IBM 1996] IBM. Intelligent Agent Resource Manager, Open Blueprint, G325-6592-0. 1996

[Lange 2000] Lange Danny, Change Daniel T. IBM Aglets Worbench Programming Mobile Agents of Java, IBM Corporation 2000.

[Lange 1998] Lange Danny. Mobile Agents: Enviroments, Technologies, and applications. Proceedings of the Practical Application of intelligent Agents and Multi-Agent Technology Conference, PAAM98. Marzo 1998.

[Lingau 1996] Lingnau Anselm, Drobnik Oswald. An infrastructure for mobile agents: requierements and arquitecture. JW Goethe

University. Frankfurt, Alemania. 1996.

[Moore 2002] MOORE, R.C. A formal theory of knowledge and action. In Readings in Planing (eds J.F. Allen, J. Hendler and A. Tate), pp. 480-519. Morgan Kaufmann, San Mateo, C.A. 2002.

[Mun 2000] Mun Choon Chan y Thomas Y.C. Woo (2000) Next-Generation Wireless Data Services: Architecture and Experience. IEEE Pers. Comm., pp. 20-33.

[MICR 2000] Microsoft Corporation. DCOM Technical Overview. USA, 2000. http://www.microsoft.com/workshop/prog/com/

[MURU 1998] Murugesan San. Intelligent agents on the internet and web: progress and prospects. Proceedings of the International Workshop on Intelligent Agents on the Internet and Web. March 1998, Mexico. pp 3-6.

[Naughton 2001] Naughton Patrick, Herbert Shildt. Java - Manual de referencia. Editorial Mc Graw Hill. España 1997 [NISH95a] Nishigaya Takashi, Design of MultiAgent Programming Libraries for Java. Fujitsu Laboratories. Japan 2001.

[Object 2000] OBJECT MANAGEMENT GROUP (OMG Inc). Mobile Agent Facility Specification, enero, 2000.

[Patriik 2000] 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.

[Peña 1999] PEÑA, C. I., MARZO, J. L. "Adaptive Intelligent Agent Approach to Guide the Web Navigation on the PLAN-G Distance Learning Platform". IEEE Colloquium "Lost in the Web - Navigation on the Internet", London, November 1999.

[Perez 2000] Pérez Arturo Jesus, (2000), Díaz SAHARA Arquitectura Integral para sistemas de agentes móviles, Universidad de Oviedo, Departamento de Informática.

[Presuman 1993] PRESSMAN, Roger S. Ingeniería del software un enfoque

practico. Traducción: RUCKAUERS C, Carlos. Tercera edición, Madrid, España, McGraw-Hill, enero, 1993

[Qusay 2001] 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.

[Self 1990] SELF, J. "Theoretical Foundations for Intelligent Tutoring Systems", Journal of Artificial Int. in Education, 1 (4), 3-14, 1990.

[Shoham 1993] SHOHAM, Y. Agent-oriented programming. Artificial Intelligence, 60:51–92, Marzo. 1993

[SUN 2005] **[SUN 2005]** Sun MicroSystem, Java ME Technology APIs & Docs (2005), http://java.sun.com/javame/reference/apis.jsp#api

[TERO 2001] Tero Ojanaperâ y Ramjee Prasad (2001) An Overview of Third-Generation Wireless Personal Comunications: A European Perspective. IEEE Pers. Comm., pp.59-65.

[Telefonica 2005] Telefónica I+D (España). (2005) Las Telecomunicasiones y la Movilidad en la Sociedad de la Informacion: AHCIT. CENTENNIAL Y CANTV, pp.23-29.

Tocher 1991] TOCHER, Alastair J. A Formal Model for Naming. ANSA, Advanced Networked Systems Architecture. Cambridge, United Kingdom, diciembre 1991

[TOMAS 1998] Tomas Baiget: General Magic, Marc Porat y los Agentes, España 1998, http://eprints.rclis.org/archive/00008380/01/General_Magic.pdf.

[WG-AdHoc 2002] Agents in AdHoc Enviroments. A Whitepaper. WG-AdHoc (2002)

[White 1996] White Jiem, Mobile Agent whit Paper, General Magic, paper 1996.

[**Wiki 2005**] Sun MicroSystem, Enciclopedia Wikipedia Foundation Inc, España 2005 http://es.wikipedia.org/wiki/Sun_Microsystems.

[Wooldridge 1999] WOOLDRIDGE, M. and JENNINGS, N. R., “Agent Theories, Architectures, and Languages: a Survey”, in Wooldridge and Jennings Eds., Intelligence Agents, Berlin: Springer- Verlag, Vol. 1, Nº 22, 1999.

[Wooldridge 1999] WOOLDRIDGE, M. and JENNINGS, N. R. Intelligent agents: Theory and practice. The Knowledge Engineering Review, 10(2):115–152, 1995.

GLOSARIO

Agente: Entidad software autónoma que se comporta según los objetivos que debe alcanzar y que reacciona ante eventos externos y puede comunicarse y colaborar con otros agentes. Con la propiedad de migrar de un sitio a otro.

SMA: (Sistema Multiagente) Ambiente de ejecución integrado por entornos, agentes y clones.

Entorno: Residencia o sitio específico donde los agentes se agrupan y desarrollan su ciclo de vida.

Tarea: Acción o modo de actuar de un agente con un propósito bien definido.

Comportamiento: Componente de un agente que permite agrupar tareas y determinar el modo de ejecución del agente.

Estado: Componente de un agente que permite conocer y gestionar el valor del momento de ejecución de un agente.

Clon: Copia de un agente con comportamientos y tareas idénticas al agente original pero con identificación diferente.

Identificación: Nombre lógico de un entorno, agente, clon, comportamiento, itinerario y tarea que le permite ser registrado y nombrado en el SMA.

Itinerario: Componente perteneciente a la estructura interna del agente, necesario para el proceso de migración.

URL: Componente básico del itinerario del agente, representa el sitio donde se realizará el viaje del agente.

Viaje: Comportamiento del agente cuando se desplaza de un sitio a otro.

Proceso: Secuencia de eventos ocurridos en una aplicación multiagente.

Migración: Proceso del ciclo de vida del agente donde viaja de un sitio a otro, ya sea dispositivo móvil o fijo. La migración comprende viajar y volver al sitio de partida del agente.

Registro: Inscribir un agente al SMA por medio de su identificación usando el nombre lógico.

Comunicación: Comportamiento del agente donde interactúa con otros agentes a través de mensajes.

Mensaje: Elemento que es transmitido entre agentes, el cual lleva información sobre procesos o rutinas de los agentes.

Canal de Comunicación: Medio utilizado por los agentes para establecer una conexión entre ellos y poder comunicarse a través de mensajes. Existe un solo canal de comunicación entre cada conexión establecida por los agentes.

Agente Emisor: Agente o clon que envía mensajes a otro u otros agentes en el proceso de comunicación.

Agente Receptor: Agente o clon que recibe los mensajes en el proceso de comunicación.

Directorio de búsqueda: Listado de agentes y clones registrados en el SMA, cada elemento registrado se compone de su localización en el SMA y su nombre lógico.

Ruta de Trabajo: Listado de los sitios adicionados en el itinerario para el proceso de migración.