Tecnologías para el Desarrollo de Sistemas Multiagentes

Marcos H. Rivas Peña[1], Giovanna M. Valverde Ayala[2]

[1, 2]Universidad Nacional Mayor de San Marcos,
Facultad de Ingeniería de Sistemas e Informática
Departamento de Ciencia de la Computación

[1]mrivasp@unmsm.edu.pe, [3]Gmvalverde@yahoo.com

RESUMEN

La tendencia a desarrollar sistemas distribuidos en ambientes dinámicos y heterogéneos requiere de tecnologías que varían en capacidades y conectividad. En respuesta a esta tendencia, la computación basada en agentes se ha convertido en el centro de desarrollo, produciendo de este modo tecnologías clave para la construcción de estos sistemas. En el presente trabajo se presenta un estudio de los middleware como infraestructura básica para facilitar el desarrollo de SMA, además de los estándares que existen para el desarrollo de esta clase de sistemas y, siguiendo la tendencia al uso de plataformas de desarrollo de agentes, se hace una descripción de algunas plataformas que han implementado las especificaciones FIPA.

Palabras clave: Sistemas Multiagente, Middleware, Estándares, Plataformas de Desarrollo.

ABSTRACT

The tendency to develop systems distributed in dynamic and heterogenous environments requires of technologies that vary in capacities and connectivity. In answer to this tendency the computation based on agents has become in center of development, producing of this way technologies nails for the construction of these systems. In the present work a study of middleware like basic infrastructure to facilitate the development of SMA, also of the standards appears that exist for the development of this class of systems and, following the tendency the use of development platforms of agents, a description becomes of some platforms that have implemented specifications FIPA.

Keywords: Multiagente Systems, Middleware, Standards, Development Platforms.

1. INTRODUCCIÓN

Un sistema multiagente (SMA) es un sistema de software que agrupa tecnologías de distintas áreas de conocimiento: técnicas de ingeniería del software para estructurar el proceso de desarrollo, técnicas de inteligencia artificial para dotar a los programas de capacidad para tratar situaciones imprevistas y tomar decisiones, y programación concurrente y distribuida para tratar la coordinación de tareas ejecutadas en diferentes computadoras bajo diferentes políticas de planificación. Debido a esta combinación de tecnologías, el desarrollo de SMA goza de cierta complejidad. Esta reunión de conceptos y tecnologías hace posible dar respuesta a problemas complejos a través de la construcción de sistemas autoorganizados y dinámicos, los mismos que pueden integrarse entre sistemas heterogéneos.

La necesidad de desarrollar aplicaciones complejas, compuestas de multitud de subsistemas que interaccionan entre sí, obliga a distribuir la inteligencia entre diversos Agentes, además de construir SMA que permitan la gestión inteligente de un sistema complejo, coordinando los distintos subsistemas que lo componen e integrando los objetivos particulares de cada subsistema en un objetivo común. Estos tipos de sistemas se emplean en muchos casos: por ejemplo, cuando los problemas son físicamente distribuidos, cuando la complejidad de la solución requiere de experiencia muy heterogénea o cuando el problema a resolver está definido sobre una red de computadoras. En realidad, la complejidad de la mayor parte de los problemas que nos encontramos hoy en día es tal que se requiere una solución distribuida capaz de adaptarse a cambios en la estructura y en el entorno, así como una metodología de desarrollo que permita la construcción de todo un sistema a partir de distintas unidades autónomas [2].

Para desarrollar SMA se distinguen dos filosofías. La primera, ve el SMA como el resultado de utilizar un lenguaje de especificación de agentes, el cual parte de principios basados en modelos operacionales y modelos formales de SMA. La segunda, estudia el SMA como un sistema software que hay que construir, el desarrollo no parte de cero sino que utiliza plataformas de desarrollo de agentes que proporcionan servicios básicos de comunicación, gestión de agentes y una arquitectura de agente. Hoy en día el desarrollo de SMA es más proclive a la utilización de plataformas de desarrollo que a la aplicación de lenguajes de agentes [5]. Para la construcción de SMA se necesita una apropiada infraestructura que nos permita un rápido desarrollo de las aplicaciones, esta infraestructura se ha desarrollado en tres campos diferentes [36], tecnologías middleware, agentes móviles y agentes inteligentes.

Como ocurre con toda nueva tecnología, uno de los problemas que hay que resolver para facilitar su aplicabilidad es la interoperabilidad (facilidad de interconexión e integración de sistemas basados en dicha tecnología) y la apertura (posibilidad de extensión). Para ello, es importante disponer de estándares y en el caso de los agentes la organización que más ha trabajado en este sentido es FIPA (Foundation for Intelligent Physical Agents) [3].

En este artículo realizaremos un estudio de las tecnologías necesarias para implantar una infraestructura que nos permita desarrollar y ejecutar SMA en ambientes empresariales. El resto del artículo tiene la siguiente organización: en el punto 2, se presenta el estado del arte de las tecnologías que facilitan el desarrollo de SMA. En el punto 3, se presentan los distintos trabajos relacionados con el estudio de tecnologías para SMA. En el punto 4, se realiza una descripción de las principales plataformas de desarrollo de SMA. Finalmente, en el punto 5, se presentan las conclusiones.

2. ESTADO DEL ARTE

Para el desarrollo de SMA se puede adoptar el uso de lenguajes de agentes o el de plataforma de desarrollo de agentes [40]. El uso de lenguaje implica un buen nivel de conocimientos y se necesita: lenguaje de programación para sistemas basado en agente, lenguaje de comunicación y coordinación entre agentes, y un lenguaje para especificación de ontologías. Una descripción de los lenguajes existentes se encuentra en [8]. En este artículo nos concentraremos en el uso de plataformas de desarrollo.

En la última mitad de la década de los años ochenta, los diseñadores de software introdujeron las plataformas middleware para ofrecer un soporte a los sistemas de software distribuido.

Los SMA se desarrollan sobre middleware y proporcionan un nuevo nivel de abstracción más intuitivo [35]. Una definición de middleware que tomaremos en cuenta para el presente trabajo es la que aparece en [1], donde se define este concepto como la capa de software que se ejecuta encima de sistemas operativos y sistemas de comunicaciones heterogéneos, ofreciendo una interfaz uniforme a las aplicaciones distribuidas.

En [36], se han identificado tres niveles de infraestructura para el desarrollo y operación de SMA: la tecnología Middleware y los agentes móviles están dirigidos a superar los problemas relacionados con la heterogeneidad, las limitaciones del host y los problemas de arquitectura de red; mientras que los agentes inteligentes están enfocados a permitir la realización de las tareas más complejas. Como tal, es útil considerar la infraestructura de los SMA en forma de capas, empezando con el middleware, preocupándonos por los agentes móviles, y subiendo a los problemas del agente inteligente. En cada nivel se han desarrollado los servicios apropiados.

2.1. Middleware

Una de las primeras tecnologías middlewares y ampliamente usada es de the Object Management Group's (OMG) CORBA [33], desarrollado con la intención de resolver los problemas de heterogeneidad de los sistemas distribuidos. CORBA es una aproximación completamente basada en objetos, ya que en el OMG se ha tenido siempre el convencimiento de que la tecnología de objetos es la más adecuada para el desarrollo de aplicaciones distribuidas porque simplifica el problema.

El desafío relacionado a garantizar la creación de aplicaciones que operen en un ambiente dinámico y heterogéneo, donde hay una variedad de dispositivos conectados con diferentes capacidades y donde existe la necesidad de solucionar problemas de coordinación en estos dispositivos para que los componentes software puedan trabajar en alcanzar objetivos comunes, está dado por la computación basada en agente, la misma que es sugerida como un paradigma que puede proveer los fundamentos conceptuales que permitan desarrollar [32] aplicaciones con trato efectivo a estos problemas.

Con la finalidad de conseguir que los sistemas basados en agente sea una corriente que prevalezca, se han desarrollado un buen número de herramientas para proveer el soporte necesario para el desarrollo de estos sistemas. Pero, a pesar del esfuerzo, las herramientas han fallado y no se han convertido en el eje principal de desarrollo, esto debido en parte a la inherente complejidad en el desarrollo de los SMA [29] y, por otro lado, debido a la falta de tecnologías bien establecidas por tratar con los problemas de infraestructura de más bajo nivel como el descubrimiento dinámico de servicios y comunicación entre agentes [38].

Sin embargo, en los últimos años nueva tecnología han emergido [27], [28], [31], entre ellas: Jini [10], JXTA [11], Universal Plug'n'Play [13], Salutation [12] y, más recientemente, Web Services [14]. Se espera que tengan el potencial para proveer la piezas faltantes en lo que se refiere a la infraestructura de bajo nivel requerida para los SMA.

Jini (Java Intelligent Network Infraestructure)

Jini fue introducido en el año 1999 como una tecnología de conectividad de red capaz de soportar redes dinámicas y heterogéneas, [25] es una arquitectura distribuida orientada a servicios desarrollada por Sun Microsystems. Su principal objetivo es convertir a la red en una herramienta flexible y fácilmente administrable en la que los clientes puedan encontrar servicios de un modo robusto y flexible. Se apoya fuertemente en Java, de tal forma que es necesario que todos los dispositivos tengan una máquina virtual Java, o un dispositivo que la tenga y actúe en su nombre.

En la arquitectura Jini se definen tres componentes: los servicios, los clientes y los servicios de directorio que se denominan Jini Lookup Services (JLS). Un servicio en Jini está representado por un objeto Java y se define como una entidad software que proporciona algún tipo de control sobre un dispositivo. Un cliente es aquel que hace uso de un servicio y, por lo tanto, un servicio puede ser a su vez cliente de otro.

El JLS es obligatorio dentro de la arquitectura de Jini, los clientes y los servicios siempre se descubren a través de él y nunca de forma directa. Para registrar o buscar un servicio primero es necesario localizar algún JLS, para

localizar un JLS Jini usa el discovery protocol y para unirse a él usa el join protocol que provee una serie de pasos estándar para iniciar y registrarse en el JSL. Para localizar un JLS se han definido tres tipos de protocolos [4] [10]: unicast discovery protocol, empleado cuando ya se conoce un JLS; multicast request protocol, empleado para buscar un JLS; y multicast announcement protocol, empleado por los JLS para anunciar su disponibilidad. Para soportar el dinamismo de la red y para que los servicios registrados en el JLS no se queden obsoletos, Jini implementa un mecanismo de leasing que obliga a que los servicios actualicen su registro periódicamente para mantener su entrada en un JLS y, de este modo, poder ser localizados por los clientes.

En Jini se define no sólo un mecanismo de descubrimiento de servicios, sino también un mecanismo de uso: un cliente le solicita a un JLS la búsqueda de un servicio proporcionándole un identificador del servicio, un interfaz Java o un conjunto de atributos, el JLS responde a esta búsqueda con un conjunto de objetos proxies que permitirán al cliente acceder a implementaciones de dicho servicio.

El mecanismo de comunicación entre clientes y servicios se basa en Java Remote Method Invocation (RMI). En cuanto a seguridad, Jini depende del modelo de seguridad de Java.

JXTA

El proyecto JXTA [11] es una plataforma de computación distribuida peer-to- peer (P2P) concebida y promovida inicialmente por Sun Microsystems a principios de 2001, pero en la que en la actualidad participan un gran número de centros de investigación académicos e industriales. JXTA proporciona una serie de tecnologías middleware por encima de las cuales se pueden construir servicios y aplicaciones. El objetivo principal del proyecto es crear un soporte software para sistemas P2P que permita la interoperabilidad entre diferentes implementaciones de un mismo servicio, la independencia de la plataforma, del lenguaje y del entorno de programación en el que se desarrolla el servicio y la ubicuidad de los servicios, permitiendo incluir en el sistema desde los grandes servidores hasta los dispositivos más limitados como PDAs.

JXTA define los peers como entidades que pueden comunicarse, para los cual ha definido seis protocolos básicos [6], entre ellos se encuentra: el Peer Discovery Protocol, que es utilizado por los peers para anunciar sus propios recursos y descubrir recursos publicados por otros peers; el Peer Information Protocol, que permite a los peers obtener información del estado de otros peers (tiempo, estado, tráfico, etc.); el Pipe Binding Protocol, que es utilizado para establecer canales de comunicación virtuales o tuberías entre uno o más peers; el Peer Resolver Protocol, a través de este protocolo los peers pueden enviar consultas genéricas a uno o más peers y recibir una respuestas a dicha consulta; el Endpoint Routing Protocol, utilizado para encontrar rutas a otros peers; y el Rendezvous Protocol, que es utilizado para propagar mensajes dentro de un peer group.

El Peer Discovery Protocol es el protocolo de descubrimiento por defecto que debe implementarse obligatoriamente en todos los sistemas JXTA.

En cuanto a la seguridad, la tecnología JXTA soporta un conjunto básico de algoritmos criptográficos utilizando una versión modificada del paquete java.security. Los servicios proporcionados son integridad y privacidad de mensajes, autenticación y protección de denegación de servicio.

UPnP (Universal Plug'n'Play)

Universal Plug-and-Play (UPnP) [13] es una arquitectura software abierta y distribuida propuesta para el descubrimiento, anuncio y uso de servicios en entornos dinámicos centrada en estandarizar protocolos de comunicación basados en XML, para la interacción entre los diferentes elementos de la arquitectura. Ha sido diseñado de forma que sea independiente del fabricante, del sistema operativo, del lenguaje de programación de cada dispositivo u computador, y del medio físico usado para implementar la red. El UPnP Forum [13] es promotor de esta iniciativa, que está liderada por Microsoft.

Simple Service Discovery Protocol (SSDP) [9] es el protocolo que se ha definido dentro de UPnP para el descubrimiento dinámico de servicios. SSDP puede operar con o sin un elemento central, denominado Service Directory en la red. Este protocolo es capaz de descubrir cuando se

conecta un nuevo equipo o dispositivo a la red, asignándole una dirección IP, un nombre lógico, informando a los demás de sus funciones y capacidad de procesamiento, e informarle, a su vez, de las funciones y prestaciones de los demás. De esta forma, el usuario no tiene que preocuparse de configurar la red ni de perder el tiempo instalando drivers o controladores de dispositivos ya que el UPnP se encarga todos estos procesos cada vez que se conecta o se desconecta un equipo y, además, optimiza en todo momento la configuración de los equipos. En UPnP no se aborda la seguridad a nivel SSDP.

Salutation

Salutation es una arquitectura para el descubrimiento y anuncio de servicios desarrollada por el Salutation Consortium [12], agrupación de compañías y centros académicos, que surge con el objetivo de resolver el problema del descubrimiento y acceso a servicios entre un amplio conjunto de dispositivos y equipos en un entorno cambiante, independientemente del protocolo de transporte concreto que se esté utilizando [9].

Salutation [30] es un estándar abierto independiente de sistemas operativos, protocolos de comunicaciones y plataformas hardware. La arquitectura Salutation define tres componentes:

- Functional Units, que desde el punto de vista de un cliente definen un servicio. Para alguno de los servicios más habituales, como impresoras, faxes o almacenamiento de documentos, el consorcio está definiendo estas unidades de forma que se garantice la interoperabilidad.
- Salutation Managers (SLMs), que permiten a los clientes descubrir y comunicarse con los diferentes servicios proporcionados en la red. El proceso de descubrimiento de servicios puede realizarse a través de múltiples SLMs. Un SLM puede descubrir otros SLMs remotos y determinar los servicios que están registrados allí. De esta manera, el descubrimiento de servicios se realiza de una manera mucho más rápida.
- Transport Managers (TMs), que permiten aislar al SLM del protocolo de transporte que se está empleando para acceder a él. De esta forma, para dar soporte a un nuevo protocolo de transporte sólo es necesario implementar un nuevo TM, sin modificar la implementación del SLM.

Salutation no sólo define mecanismos de descubrimiento, sino que en la especificación también se describen mecanismos para el acceso a los servicios. Este soporte se proporciona a través del SLM que tiene tres modos de operación: native personality, que sólo se emplea para descubrir servicios y para establecer la comunicación entre los clientes y el servidor; emulated personality, que además de establecer la conexión sirve como pasarela entre el protocolo que emplea el cliente y el que emplea el servidor cuando son diferentes; y salutation personality, en el que además de establecer la conexión obliga a que la comunicación entre cliente y servidor se realice en un formato determinado.

En cuanto a seguridad, Salutation soporta autenticación de usuario mediante esquemas de identificador y password. Opcionalmente, un servicio puede obtener información registrada de los sistemas que están accediendo a él.

Web Services

Un servicio Web es una colección de protocolos y estándares que sirve para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladasen lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma pueden utilizar los servicios web para intercambiar datos en redes de computadoras como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Los mensajes para invocar el servicio se codifican en The Extensible Markup Language (XML) [15] y se pueden transportar utilizando HTTP. La Organization for the Advancement of Structured Information Standards (OASIS) [16] y el World Wide Web Consortium (W3C) [14] son las organizaciones responsables de la arquitectura y reglamentación de los servicios Web. La infraestructura básica de los Web Services se pueden definir: (1) en términos de lo que va en la red, aquí se usa XML como formato para los datos que se intercambian y un protocolo independiente del sistema operativo y del lenguaje de programación SOAP, usado para el intercambio de información; (2) en una descripción de los servicios en la red, usando un lenguaje de descripción WSDL (Web Services

Description Languaje), derivado de XML; y (3) en lo que nos permite encontrar y almacenar servicios, usando UDDI (Universal Description, Discovery and Integration), todo esto nos permite integrar servicios formando un directorio distribuido de servicios, almacenar WSDL y datos complementario del proveedor y localizar servicios.

2.2. Estándar para Construcción de SMA

Las herramientas de desarrollo de SMA deben poseer una serie de características que heredarán las aplicaciones creadas y que harán que esas aplicaciones sean más o menos eficientes para aplicarlas a un dominio determinado. Además, en toda nueva tecnología, uno de los problemas que hay que resolver para facilitar su aplicabilidad es la interoperabilidad (facilidad de interconexión e integración de sistemas basados en dicha tecnología) y la apertura (posibilidad de extensión). Para ello es importante disponer de estándares y en el caso de los agentes las organizaciones que han trabajado en esta dirección son la OMG [17] y FIPA (Foundation for Intelligent Physical Agents) [18].

El grupo OMG (Object Manager Group) ha desarrollado el estándar MASIF (Mobile Agent System Interoperabilities Facility), MASIF propone el desarrollo de sistemas de agentes cuyos entornos se basan en una composición entre agentes (componentes) y agencias (lugares), entidades que colaboran utilizando patrones y políticas de interacción generales. En este modelo, los agentes se caracterizan por sus capacidades tipos de interacciones y, sobre todo, por la movilidad. Por otro lado, las agencias, soportan la ejecución concurrente de los agentes, y proporcionan seguridad y movilidad entre otras cosas. Es por tanto un estándar que facilita la creación de sistemas de agentes móviles.

La fundación FIPA es la organización que más ha trabajado en los estándares, entre los documentos más importantes que publica esta organización podemos encontrar el que define la Arquitectura Abstracta FIPA [3]. Su objetivo es, tal y como se expresa en el documento, el permitir la construcción de sistemas que se integren con su entorno de computación particular, mientras que interoperan con sistemas de agentes que residen en entornos heterogéneos, todo con el mínimo esfuerzo.

El estándar FIPA define los servicios que debe proporcionar toda plataforma de agentes: un sistema encargado del transporte de mensajes (Internal Platform Message Transport, IPMT), un sistema de gestión de agentes (Agent Management System, AMS), un servicio de directorio (Directory Facilitator, DF) y un canal de comunicaciones para los agentes (Agent Communication Chanel, ACC). Cada uno de estos servicios, excepto el de transporte de mensajes, es suministrado por agentes especializados, lo cual supone que la comunicación con ellos será mediante mensajes ACL (Agent Communication Language) con la ontología definida para ese servicio.

El AMS es el elemento de gestión principal, que conoce en todo momento el estado de su plataforma y de los agentes que pertenecen a ella. Entre los servicios que ofrece están: la creación, destrucción y control del cambio de estado de los agentes, supervisión de los permisos para que nuevos agentes se registren en la plataforma, control de la movilidad de los agentes, gestión de los recursos compartidos y gestión del canal de comunicación.

Cada AMS proporciona también un servicio de nombres (Agent Name Service-ANS), también llamado servicio de páginas blancas. Este servicio asocia el nombre que identifica a un agente en su sociedad y la dirección de transporte real en la que se encuentra.

Como complemento al Servicio de Nombres, la plataforma de agentes incluye un servicio de páginas amarillas, que permite buscar un agente por sus capacidades y no sólo por su nombre. Esta labor la realiza el DF. Los agentes se registran en él indicando los servicios que ofrecen. Cuando otro agente tiene unas necesidades concretas lanza una búsqueda del servicio deseado obteniendo los agentes que le ofrecen estos servicios.

Todos los agentes FIPA deben tener acceso a un ACC. El ACC es el elemento encargado de gestionar el envío de mensajes entre agentes de una plataforma y entre agentes de distintas plataformas.

Internal and Agent Platform Message Transport (IPMT-APMT). El IPMT representa toda la infraestructura de comunicaciones que va a permitir que dos agentes dentro de la misma pla-

taforma puedan comunicarse. El Agent Platform Message Transport (APMT) es otro elemento que hace la misma labor que el IPMT pero a nivel de comunicaciones entre plataformas.

Para los problemas de seguridad FIPA ha estudiado soluciones de seguridad para que los SMA puedan desplegarse en sistemas abiertos con las garantías requeridas. Este tema no está cerrado actualmente y es un área de investigación permanente. Las recomendaciones de FIPA están basadas en la utilización de los estándares existentes, siempre que esto sea posible.

3. TRABAJOS RELACIONADOS

Entre los trabajos encontrados relacionados con la infraestructura para la construcción de SMA tenemos.

En el año 2000, Pierre-Michel Ricordel and Y.Demazeau [34] presentan criterios para evaluar las plataformas para desarrollo de SMA, enfocado en cuatro etapas para la construcción de software: análisis, diseño, desarrollo e implementación. Desarrolla el estudio en cuatro plataformas, AgentBuilder, Jack, Madkit and Zeus.

R. Ashri and M. Luck (2000-2001) [36] [37], sostienen que los elementos requeridos para establecer una infraestructura para un rápido desarrollo de SMA, abarca tres campos diferentes de investigación: tecnologías middleware, agentes móviles y agentes inteligentes, sin embargo, estos tienen un alto grado de superposición. Presentan una infraestructura técnica para desarrollo de sistemas multiagentes, PARADIGMA, que une teoría con ejecución práctica en un intento de proveer un conjunto de herramientas para el desarrollo de SMA. Esta infraestructura está basada en SMART agent framework que provee una infraestructura conceptual diferenciando entre entidades agentes y no agentes en el ambiente.

Gerhard WeiB (2002) [8], ofrece una guía sobre la amplia literatura existente en el campo de la Ingeniería de Software Orientado a Agente, elaborada pensando en el alto valor que tiene para investigadores y desarrolladores de SMA. Las referencias se presentan en una estructura de acuerdo a los temas importantes que se deben considerar para el desarrollo de SMA: Metodologías y Framework para ingeniería de requerimientos, análisis, diseño, desarrollo y implementación; lenguajes de programación, comunicación y coordinación; especificación de ontologías; y herramientas de desarrollo y plataformas.

Ronald Ashri (2003) [38], presenta un revisión de las tecnologías middleware Jini y JXTA, analizando su conveniencia en el desarrollo de SMA, concluye que ambas presentan ventajas y ciertas dificultades en el desarrollo de las tareas, lo que significa que la decisión de su uso se debe tomar en base al tipo de aplicación que se va a desarrollar.

M. Luck, P. McBurney, O. Shebory and S. Willmott (2005) [31], intenta contribuir en el conocimiento de tecnología de agentes informando a la variada audiencia sobre el estado del arte de la tecnología y la dirección que este campo puede tomar en el futuro. Presenta una visión coherente del desarrollo en el campo de tecnología de agentes, sus áreas de aplicación y las barreras probables que se pueden encontrar.

4. PLATAFORMAS PARA DESARROLLO DE SMA

Actualmente, existe una gran cantidad de plataformas para agentes, en [19] se encuentra un lista de más de 100 herramientas para desarrollar sistemas basados en agente y en [39] se presenta una revisión de los productos software para SMA. Normalmente, todas las herramientas están implementadas en un lenguaje orientado a objetos, y este lenguaje en la mayoría de los casos es Java, aunque también hay algunas implementadas en otros lenguajes como C++.

Mencionaremos brevemente algunas herramientas con mayor publicación que han implementado su plataforma de agentes de acuerdo a las especificaciones FIPA [23].

4.1. Grasshopper

Grasshopper [24] es una plataforma para agentes móviles inteligentes 100 % abierta, basada en java y desarrollada conforme a las especificaciones de OMG MASIF y FIPA.

Proporciona dos extensiones optativas de acuerdo a cada estándar para la interoperabilidad entre agentes o entre plataformas de agentes.

Grasshopeer [7] se realiza en un ambiente distribuido del agente (DAE). El DAE se compone de regiones, de lugares, de agencias y de diversos tipos de agentes. Un lugar proporciona un grupo lógico de funcionalidades dentro de una agencia. El concepto de la región facilita la gestión de los componentes distribuidos (agencias, lugares, y agentes) en el ambiente Grasshopper. Las agencias, así como sus lugares, pueden ser asociados a una región específica colocándolos dentro del registro de la región. Todos los agentes que son recibidos actualmente por esas agencias también serán colocados automáticamente en el registro de la región.

Dos tipos de agentes son conocidos en Grasshopper, agentes móviles que tienen la capacidad de moverse de una red a otra y los agentes estacionarios que no tiene la capacidad de migración entre redes.

Los siguientes servicios son proporcionados por las agencias en Grasshopper:

- Servicio de la Comunicación: Este servicio es responsable de todas las interacciones alejadas que ocurran entre los componentes distribuidos de Grasshopper.
- Servicio de Registro: Cada agencia debe saber respecto a todos los agentes y lugares recibidos actualmente.
- Servicio de Gestión: Los servicios de gestión permiten la supervisión y el control de agentes y los lugares de una agencia de los usuarios.
- Servicios de seguridad: Existen dos mecanismos de seguridad: la externa, que protege las interacciones entre componentes distribuidos, agencias y registro de regiones; y la interna, que protege los recursos de las agencias contra el acceso desautorizado de los agentes.
- Servicio de Persistencia: Permite el almacenaje de los agentes y de los lugares (la información interna mantenida dentro de estos componentes) en un medio persistente. De esta manera, es posible recuperar agentes o lugares cuando está necesitado.

4.2. Zeus

El proyecto Zeus lo ha llevado a cabo BT Labs. (British Telecommunications Laboratories) con el objetivo de construir un conjunto de herramientas (Zeus Agent Building Toolkit) para facilitar el desarrollo de nuevos SMA. Zeus ha sido desarrollado en Java y está basado en el paradigma de la programación visual, así el proceso de creación y control de agentes está apoyado por un entorno gráfico de ventanas y menús para poder configurar las funcionalidades y características de estos.

La herramienta [21] se compone de tres grupos funcionales de software: una librería de componentes sobre la que se construyen los agentes, un conjunto de editores que permiten realizar una creación declarativa de los agentes y un conjunto de herramientas de visualización del sistema que permite visualizar el comportamiento del sistema en tiempo de ejecución.

Zeus implementa un interfaz para el desarrollo y otro para el control de los agentes durante su ejecución, no se permite movilidad de agentes en tiempo de ejecución. Los agentes pueden ser creados en un host remoto o local. Para la comunicación entre los agentes se utiliza el lenguaje de comunicación estándar de FIPA, el ACL y proporciona un directorio que facilita la localización de agentes y recursos.

4.3. Jade (Java Agent Development framework)

Jade[22] es un middleware desarrollado por Telecom Italia Lab (TILAB) para el desarrollo de SMA distribuidas basado en la arquitectura de comunicación peer-to-peer, estotalmente implementada en Java, cuyo objetivo es simplificar el desarrollo de SMA a través de un conjunto de sistemas, servicios y agentes. Intenta contemplar en sus sistemas los estándares FIPA en su totalidad, no presenta un interfaz para el desarrollo pero sí para el control de la ejecución del sistema de agentes. Utiliza como lenguaje de comunicación el estándar FIPA ACL y es posible la movilidad de agentes dentro de la plataforma. Proporciona un conjunto de herramientas (agentes) que simplifican la administración de la plataforma de agentes, las que están disponibles son las siguientes: el Dummy Agent, agente que permite enviar y recibir mensajes ACL; el DF(Facilitador de Directorios), que gestiona los

directorios de las plataformas de agentes; el Sniffer, agente que muestra los mensajes que se intercambian ciertos agentes de un contenedor; el Introspector, agente que muestra el ciclo de vida de otro agente; y el RMA, que permite lanzar, matar y gestionar los distintos agentes. De entre estas herramientas, es el agente Sniffer el que tiene mayor interés para el caso de visualización del comportamiento del sistema, ya que las otras, aunque también proporcionan ayudas de visualización, van más dirigidas a tareas de depuración que le permiten al usuario interaccionar con los agentes del sistema modificando su camino de ejecución comprobando su respuesta ante distintas situaciones.

La versión 3.3 de JADE puede bajarse de la siguiente dirección [20], lo único que necesita JADE para funcionar es una versión correcta del Java Run Time Environment.

4.4. LEAP (Lightweight Extensible Agent Platform)

El proyecto LEAP [26] está siendo desarrollado por un consorcio de compañías entre las que se encuentran, entre otras, Motorola, British Telecom y Siemens. El objetivo del proyecto es el desarrollo de una plataforma de agentes móviles conforme al estándar FIPA que pueda operar tanto en dispositivos móviles (teléfonos móviles, PDA) como en PCs. El proyecto comenzó en Enero de 2000 y tiene dos fases, la primera de ellas, ya finalizada, consistió en la revisión de los estándares FIPA y WAP y el diseño e implementación de una plataforma de agentes para dispositivos móviles. La segunda fase consiste en evaluar la plataforma en sistemas reales y analizar las prestaciones obtenidas en dos aplicaciones: asistencia en carretera y tareas de gestión de red.

5. CONCLUSIONES

Uno de los problemas principales en el desarrollo de SMA era la falta de integración entre sistemas heterogéneos. Hoy en día, con una infraestructura adecuada utilizando middleware, se está logrando esta integración en ambientes dinámicos. Por otro lado, la tendencia en la construcción de SMA es usar plataformas de desarrollo de agentes, y se ve que la mayoría de las plataformas o entornos han sido desarrollados en lenguaje Java, esto debido a la ventaja que ofrece Java de desarrollar software independiente del procesador.

Actualmente, la mayoría de las plataformas de desarrollo y ejecución de agentes tienden a adoptar, o ser compatibles, con el estándar FIPA.

Para el desarrollo de agentes existen productos que son: el software libre y productos comerciales.

REFERENCIAS

[1] Bacon, J., Moody, K., Bates, J., Hayton, R., Ma, C., McNeil, A., Seidel, O. y Spiteri, M. (2000). "Generic Support for Distributed Applications". IEEE Computer, 33(3):68-76. March, 2000.

[2] Ferber, J., (1999) Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence, Addison-Wesley.

[3] Foundation for Intelligent Physical Agents. FIPA abstract architecture specifcation. Technical report, FIPA, February 2002.

[4] Goland, Y. Y., Cai, T., Leach, P., and Gu, Y. (1999). Simple Service Discovery Protocol/1.0. Internet-Draft (work in progress). draft-cai-ssdp-v1-03.txt.

[5] Gómez J. Modelado de Sistemas Multiagente. Departamento de Sistemas Informáticos y Programación, Facultad de Informática, Universidad Complutense de Madrid, Tesis Doctoral Junio 2002.

[6] Gong, L. (2001). JXTA: A Network Programming Environment. IEEE Internet Computing, pages 88.

[7] Grasshopper (1998). Grasshopper Technical Overview.

[8] Gerhard Weib, Agent Orientation in Software Engineering, The Knowledge Engineering Review, Vol.16(4), 2001.

[9] Golden G. Richard Service Advertisement and Discovery: Enabling Universal Device Cooperation, IEEE Internet Computing Setember 2000, pp 18-26.

[10] http://www.jini.org/. Jini Technology.

[11] http://www.jxta.org/. The JXTA Project.

[12] http://www.salutation.org/. The Salutation Consortium.

[13] http://www.upnp.org/. Universal Plug n'Play.

[14] http://www.w3.org/2002/ws/. Web Services.

[15] http://www.w3.org/xml/.

[16] http://www.oasis-open.org

[17] http://www.omg.org

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

[19] http://www.agentlink.org

[20] http://jade.tilab.com

[21] http://more.btexact.com/projects/agents/zeus/

[22]http://jade.tilab.com/papers/WhitePaperJADEEXP.pdf

[23]http://www.fipa.org/resources/livesystems.html

[24] http://www.grasshopper.de/

[25] Jini (1999). Jini Architectural Overview. White Paper.

[26] LEAP Project (2000). Deliverable D31: Specications of the LEAP Architecture.

[27] M. Luck, P. McBurney and C. Preist, A Manifesto for Agent Technology: Towards Next Generation Computing, Journal of Autonomous Agents and Multi-Agent Systems, 9(3), 203-252, 2004.

[28] M. Luck, P. McBurney and C. Preist, Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing), AgentLink, 2003.

[29] M. Luck. From definition to development: What next for agent-based systems. Knowledge Engineer-ing Review, 14(2):119–124, 1999.

[30] Miller, B. A. and Pascoe, R. A. (2000). Salutation service discovery in pervasive computing environments. Technical report, IBM White Paper.

[31] M.Luck ,P.McBurney ,O. Shehory ,and S. Willmott, Agent Technology Roadmap: A Roadmap for Agent Based Computing, AgentLink, September 2005. ISBN 085432 845 9.

[32] N.R. Jennings. On Agent-Based Software Engineering. Artificial Intelligence, 117(2):277–296, 2000.

[33] OMG:CORBA v 3.0.3 Specification. http://www.omg.org.

[34] Pierre-Michel Ricordel and Y. Demazeau, From analysis to deployment: a multiagent platform survey. In A. Ominici R. Tolksdorf, and F. Zambonelli, editors, Proceedings of 1st International Workshop on Enginnering Societis in the Agents World (ESAW), ECAI`2000, pages 93-105, Berlin, Germany, November. Springer Verlag.

[35] R. Ashri, M. Luck and M. d'Inverno, Infrastructure Support for Agent-based Development, in Foundations and Applications of Multi-Agent Systems, M. d'Inverno, M. Luck, M. Fisher and C. Preist (eds.), Lecture Notes in Artificial Intelligence 2403, Springer-Verlag, 73-88, 2002.

[36] R. Ashri and M. Luck, Towards a layered approach for agent infrastructure: the right tools for the right job, in Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, 9-16, 2001.

[37] R. Ashri and M. Luck, Paradigma: Agent Implementation through Jini, in Proceedings of the Eleventh International Workshop on Database and Expert Systems Applications, A. M. Tjoa and R. R. Wagner and A. Al-Zobaidie (eds.), 453-457, IEEE Computer Society, 2000.

[38] R. Ashri Agent Middleware Technologies: A review of Jini and JXTA, KTWeb.org, 2003.

[39] Review of Software Products for multi-agent systems, Agentlink, june 2002.

[40] Weyns, D., Parunak, H.V.D., Michel, F., Holvoet, T., Ferber, J.: Environments for multiagent systems: State-of-the-art and research challenges. Volume 3477 of Lecture Note in Artificial Intelligence LNAI., Springer,2005.