Plataformas para Desarrollo de Sistemas Multiagente. Un Análisis Comparativo

Tulio José Marchetti
tjm@cs.uns.edu.ar

Alejandro Javier García
agarcia@cs.uns.edu.ar

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Avda. Alem 1253 - (8000) Bahía Blanca
Tel: ++54 291 4595135 - Fax: ++54 291 4595136

El objetivo de esta línea de trabajo es el desarrollo de una plataforma para la construcción de sistemas multiagente. Como parte de esta investigación se desean analizar plataformas ya existentes en la literatura. En particular en este trabajo se presentan y evalúan las cinco plataformas que se nombran a continuación. JACK [8] la cual provee un entorno de desarrollo orientado a agentes construido sobre Java y completamente integrado con este lenguaje de programación. JADE (Java Agent Development Framework) [9, 2] que ofrece un entorno que simplifica la implementación de sistemas multiagente mediante una capa de soporte (middle-ware). JAFMAS (Java Framework for Multi-agent Systems) [10, 3] que provee una metodología genérica para desarrollar sistemas multiagente basados en los actos del habla. MADKit (Multi-agent Development Kit) [12, 5] que ofrece una plataforma multiagente para desarrollar y ejecutar aplicaciones basadas en un paradigma orientado a la organización. Y finalmente el proyecto ZEUS [15, 4] cuyo objetivo es proveer una herramienta de propósito general y personalizable, que pueda ser usada por ingenieros de software con poca experiencia en tecnología de agentes para crear sistemas multiagente.

Comparación de las plataformas

Las plataformas que hemos elegido para comparar provienen de diferentes fuentes. Dichas fuentes provienen de variados laboratorios, líneas de investigación y trabajo de diversas universidades. Sin embargo todas las plataformas consideradas en este trabajo, comparten la característica de estar implementadas en Java, y puede asegurarse su funcionamiento en Java 1.3.

Para poder comparar las plataformas mencionadad se eligireron diversos criterios que abarcan diferentes aspectos de las mismas y no incluyen sólo a su funcionamiento. Los detalles sobre cada una de las plataformas serán presentados en las próximas secciones.

Los criterios elegidos para realizar la comparación son: si la plataforma soporta algún lenguaje de comunicación entre agentes (ACL), ya sea FIPA ACL [7] y/o KQML [6], si soporta movilidad de código, cuál es la arquitectura base de la plataforma, de que tipo son los agentes soportados, cuáles son los lenguajes en los que se pueden desarrollar los agentes, cuales son las licencias de los paquetes de software, si están disponiles en Internet, cuán dificultosa es la instalación de dichos paquetes, que tan completa es la documentación y que tipo de interface posee.

A continuación describiremos con más detalle cada una de las plataformas comparadas. Los resultados de la evaluacion de estas plataformas bajo los criterios antes expuestos se encuentran en la siguiente tabla.

	JACK	JADE	JAFMAS	MADKit	ZEUS
ACL soportado	KQML FIPA ACL	FIPA ACL	KQML FIPA ACL	KQML	KQML
Movilidad de código	No detalla	No detalla	Sí	No detalla	No detalla
Arquitectura base	BDI	JVM Java RMI	No detalla	Agente/ grupo/rol	BDI[1]
Tipo de agentes soportados	BDI	Cualquiera	Cualquiera	Cualquiera	Deliberat. Colaborat.
Lenguajes soportados para implementación de agentes	Jack	Java Jess [11]	Java	Java Jess [11] Python [13] Scheme [14] BeanShell [1]	Java
Licencias del software	Beta release	LGPL	Free	GPL/LGPL	Mozilla Public
Disponibilidad on-line	Si	Sí	Sí	Sí	No[2]
Instalación	Simple	Simple	Simple	Simple	Desconocido
Documentación	Muy completa	Muy completa	Completa	Muy completa	Completa
Interface	GUI	GUI	GUI	GUI	GUI

JACK

JACK [8] provee un entorno de desarrollo orientado a agentes construido sobre Java y completamente integrado con este lenguaje de programación. Incluye todas las componentes del entorno de desarrollo de Java así como también ofrece extensiones específicas para implementar el comportamiento de los agentes. La relación entre JACK y Java es análoga a la relación entre los lenguajes C++ y C, JACK fue desarrollado para proveer extensiones orientadas a agentes al lenguaje Java. El código JACK es primero compilado a código Java regular antes de ser ejecutado.

El lenguaje JACK Agent hace más que extender la funcionalidad de Java - provee un entorno para soportar un nuevo paradigma de programación. Este lenguaje es orientado a los agentes y es utilizado para implementar sistemas de software orientados a agentes. Todas las formas en las que extiende a Java, son implementadas como plug-ins, lo que permite que el lenguaje sea lo más extensible y flexible posible. Les extensiones son las siguientes:

- define nuevas clases base, interfaces y métodos
- provee extensiones a la sintaxis de Java para soportar clases orientadas a agentes
- provee extensiones semánticas para soportar la ejecución del modelo

JADE (Java Agent Development Framework)

JADE [9, 2] es un entorno que simplifica la implementación de sistemas multiagente mediante una capa de soporte (middle-ware) que respeta las especificaciones FIPA [7] y con un conjunto de herramientas para el desarrollo y debugging. La plataforma puede ser distribuida en varias máquinas

[1] No está explícitamente indicado, pero responde a esta arquitectura.

[2] Existe una página para bajar el software pero no funciona correctamente.

(las cuales no necesitan compartir el mismo sistema operativo) y la configuración puede ser controlada mediante una interface gráfica remota. La configuración puede incluso ser cambiada en tiempo de ejecución moviendo agentes de una máquina a otra, cuando es necesario.

La arquitectura de comunicación ofrece mensajes flexibles y eficientes, mientras que JADE crea y maneja una cola de mensajes ACL entrantes; los agentes pueden acceder su cola mediante una combinación de varios modos: blocking, polling, timeout y pattern matching. El modelo de comunicación de FIPA ha sido implementado completo y sus componentes han sido claramente distinguidas y completamente integradas: protocolos de interacción, ACL, lenguaje de contenido, esquemas de codificación, ontologías y finalmente protocolos de transporte. El mecanismo de transporte, en particular, es como un camaleón debido a que se adapta a cada situación, seleccionando transparentemente el mejor protocolo disponible entre RMI de Java, notificación de eventos e IIOP. Otros protocolos pueden ser fácilmente agregados. Integraciones de SMTP, HTTP y WAP ya están planificadas para ser incorporadas. Muchos de los protocolos de interacción de FIPA ya están disponibles y pueden ser instanciados después de definir la dependencia de la aplicación de cada estado del protocolo. SL y ontologías de manejo de agentes están implementados, así como el soporte para lenguajes de contenido y ontologías definidos por el usuario que pueden ser implementadas, registradas con los agentes y automáticamente utilizadas por el framework.

JAFMAS (Java Framework for Multi-agent Systems)

JAFMAS [10, 3] provee una metodología genérica para desarrollar sistemas multiagente basados en los actos del habla junto con un conjunto de clases para soportar la implementación de estos agentes en Java. La intención del framework es asistir a los desarrolladores principiantes y expertos a estructurar sus ideas en aplicaciones de agentes concretas. La metodología esta basada en cinco etapas:

1. Identificación de agentes
2. Definición de las conversaciones de cada agente
3. Determinación de las reglas que gobiernan las conversaciones de cada agente
4. Analizar la coherencia entre todas las conversaciones en el sistema
5. Implementación

El soporte para la comunicación está provisto para ambos casos de comunicación, directo y broadcast basado en sujetos. El soporte lingüista es provisto por los lenguajes de comunicación basados en los actos del habla (Ej. : KQML [6]). El soporte de coordinación proviene de la conceptualización de los planes de un agente y su coordinación como conversaciones basadas en reglas representadas mediante modelos de autómatas.

MADKit (Multi-agent Development Kit)

MADKit [12, 5] es una plataforma multiagente para desarrollar y ejecutar aplicaciones basadas en un paradigma orientado a la organización. Estos paradigmas multiagente utilizan agentes,grupos y roles como los puntos bases para construir aplicaciones complejas. MADKit no fuerza ninguna consideración acerca de la estructura interna de los agentes, de esta manera permite a los desarrolladores implementar libremente sus propias arquitecturas de agentes.

MADKit es también una plataforma distribuida que permite el desarrollo de eficientes aplicaciones distribuidas. Para los programadores, todas las consideraciones acerca de componentes distribuidos básicos tales como “sockets” y “ports”, son totalmente transparentes. Una aplicación desarrollada en una manera multiagente puede ser ejecutada de una manera distribuida sin cambiar ninguna línea

de código. Los mecanismos de distribución de MADKit no utilizan las técnicas bastante lentas de RMI o CORBA de acceso remoto, de esta manera permite un modo eficiente de comunicación.

MADKit está construido alrededor del concepto de micro-kernel y agentificación de servicios. El kernel de MADKit es más bien pequeño, pero los agentes ofrecen los servicios importantes que se pueden necesitar para las aplicaciones. Distribución y pasaje remoto de mensajes, monitoreo y observación de agentes, edición, etc... son todas realizadas por agentes. Es más, MadKit provee un conjunto de "contenedores", esto es entornos de ejecución para correr aplicaciones, con el objetivo de que MadKit trabaje en diferentes situaciones: como un entorno de desarrollo, pero también como una herramienta embebida para aplicaciones.

ZEUS

El objetivo del proyecto ZEUS [15, 4] es facilitar el desarrollo rápido de nuevas aplicaciones multiagente mediante la abstracción de los principios y componentes más comunes a una herramienta. La idea es proveer una herramienta de propósito general y personalizable, que permita la creación de agentes colaborativos y que pueda ser usada por ingenieros de software con poca experiencia en tecnología de agentes para crear sistemas multiagente. Para esto es necesario cumplir con los siguientes principios:

- La herramienta debe permitir separar el problema de nivel de dominio y la funcionalidad de nivel de agente
- La herramienta debe estar basada en el paradigma de programación visual
- La herramienta debe soportar un diseño abierto para asegurar que sea fácilmente extensible
- Se debe utilizar tecnología "estandarizada" siempre que sea posible

Los agentes deben ser deliberativos en el sentido de que deben explícitamente razonar acerca de sus acciones en términos de que metas seguir, cuando considerar nuevas metas y cuando abandonar metas existentes. Es más, el requerimiento del comportamiento dirigido por metas implica que el agente solo seleccionará acciones que espera de alguna manera lo acerque a la meta deseada. Sólo se descartan metas cuando, no son alcanzables o las motivaciones para alcanzar dicha meta ya no se verifican.

Conclusiones

Del análisis realizado hasta el momento puede observarse que todas las plataformas tienen la misma funcionalidad básica. Sin embargo, todas poseen alguna característica que la diferencia de las demás, permitiéndole adaptarse mejor a situaciones particulares. Por ejemplo, algunas permiten desarrollar agentes en un lenguaje distinto de Java, otras permiten utilizar un lenguaje de comunicación entre agentes determinado, otras permiten movilidad de código, otras definir el tipo de agente que compone el sistema, etc.

En el caso JADE, es importante destacar que esta plataforma cumple con las especificaciones de un estándar internacional como es FIPA. Esto resulta de suma utilidad, entre otras cosas, cuando se requiere migrar sistemas ya desarrollados con este estándar. Por otro lado, cabe aclarar que la arquitectura base "JVM - Java RMI" indicada en el cuadro comparativo corresponde a la arquitectura de la plataforma, y no a una arquitectura de agentes.

Una característica importante de MADKit, es que además de permitir la creación y ejecución de los sistemas creados mediante esta plataforma, permite integrarlos en otros sistemas. De esta manera, los sistemas multiagente creados con esta plataforma funcionarán embebidos dentro de otro sistema que lo utiliza como una componente.

Aunque la documentación de ZEUS no dice explícitamente que soporte la arquitectura BDI (Beliefs, Desires and Intentions), la forma en que se describen los agentes sigue los lineamientos de una arquitectura conestas características. Esto es importante ya que la arquitectura BDI se ha convertido en una de las mas utilizadas para la especificación de agentes deliberativos.

Como se dijo antes, el objetivo de esta línea trabajo es el desarrollode una plataforma para la construcción de sistemas multiagente. Por lo tanto, se planea explorar en profundidad estas plataformas, utilizando diversos ejemplos de aplicación, a fin de descubrir los elementos esenciales que debe proveer todo entorno de desarrollo de sistemas multiagente, y que aspectos importantes pueden no haber sido incluidos en las plataformas evaluadas. Esta investigación es esencial para lograr el objetivo antes propuesto.

Referencias

[1] BEANSHELL. Lenguaje beanshell. In *http://www.beanshell.org/*, 2003.

[2] Poggi Bellifemine and Rimassa. Developing multi-agent systems with jade. *Seventh International Workshop on Agent Theories, Architectures, and Languages*, 2000.

[3] Deepika Chauhan. *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation*. PhD thesis, ECECS Department Thesis, University of Cincinnati, Cincinnati, OH, EUA, December 1997.

[4] Jaron Collins and Divine Ndumu. The zeus agent building toolkit. In *ZEUS Technical Manual*, 1999.

[5] J. Ferber and O. Gutkenecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS 98)*. IEEE CS Press, 1998.

[6] Tim Finin and Yanis Labrou. Kqml as an agent comunication language. In *Software Agents*. MIT Press, 1997.

[7] FIPA. Foundation for intelligent physical agents. In *http://www.fipa.org/*, 2002.

[8] JACK. Jack intelligent agents. In *http://www.agent-software.com/*. Agent Oriented Software Pty. Ltd., 2003.

[9] JADE. Java agent development framwork. In *http://www.sharon.cselt.it/projects/jade/*. TILAB, 2002.

[10] JAFMAS. Java-based framework for multi-agent systems. In *http://www.ececs.uc.edu/ abaker/JAFMAS*. University of Cincinnati, 2002.

[11] JESS. Lenguaje jess. In *http://herzberg.ca.sandia.gov/jess/*. SANDIA, 2003.

[12] MADKIT. Multi-agent development kit. In *http://www.madkit.org*, 2002.

[13] PYTHON. Lenguaje de script phyton. In *http://www.jython.org/*. JYTHON, 2003.

[14] SCHEME. Lenguaje de script scheme. In *http://www.gnu.org/software/kawa/*. KAWA, 2003.

[15] ZEUS. The zeus agent building tool. In *http://more.btexact.com/projects/agents/zeus/*. British Telecommunications, 2002.