Evaluación de Plataformas para el Desarrollo de Sistemas Multiagente.

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

Alejandro Javier García
ajg@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

Resumen

El objetivo de este trabajo es evaluar tres plataformas para el desarrollo de sistemas multiagente con el propósito de definir, de manera general, elementos de juicio que puedan tenerse en cuenta al momento elegir una plataforma para desarrollar aplicaciones. Para esto, se han elegido tres plataformas que se encuentran disponibles en internet y que a nuestro entender resultan como las más prometedoras. Lamentablemente, no se ha encontrado en la literatura ninguna especificación de qué criterios deben utilizarse para comparar este tipo de plataformas. Por lo tanto, en este trabajo se proponen algunos criterios que permiten realizar una comparación general, y que podrían utilizarse para extender la comparación a otras plataformas. Las plataformas elegidas son JACK, MADKit y ZEUS.

Palabras clave: Sistemas Multiagente, Diseño Orientado a Agentes, Plataformas de Desarrollo

1. Introducción

Los sistemas multiagente constituyen un área de continuo crecimiento, con resultados promisorios en diversas áreas de aplicación como inteligencia artificial, sistemas distribuidos, simulación, etc. Estos sistemas proveen un modelo atractivo para el desarrollo de aplicaciones, y las áreas en donde se los utiliza crecen día a día. Su uso ya no es exclusivo de investigadores o expertos en inteligencia artificial, actualmente programadores, diseñadores y usuarios en general se han volcado a utilizar esta metodología.

En general la implementacíón de aplicaciones con sistemas multiagente ha sido realizada por expertos donde los agentes, la plataforma de comunicación y el protocolo de interacción han sido programados en forma ad-hoc para cada sistema. A pesar que estas aplicaciones se basan en los mismos conceptos teóricos de agentes y sistemas multiagente, al momento de implementarlas

*Miembro del Instituto de Investigación en Ciencia y Tecnología Informática (IICyTI)

se han utilizado diferentes lenguajes de programación de propósito general, y cada aplicación a resuelto el mismo problema de diferente manera. Por ejemplo, el protocolo de comunicación entre agentes ha sido resuelto de diferentes maneras en la mayoría de las aplicaciones.

Actualmente, existe acuerdo en la comunidad de investigadores de sistemas multiagente que es necesario disponer de estándares para la implementación de aplicaciones. Tal es el caso de lenguajes cómo KQML [10], o los estándares definidos por la organización FIPA [11]. Sin embargo, estos estándares no son suficientes para permitir que un amplio espectro de programadores o usuarios pueda desarrollar su propia aplicación utilizando un sistema multiagente. Para resolver este problema, en los últimos dos años, se han desarrollado plataformas que facilitan el diseño e implementación de sistemas multiagente, acercando esta metodología a un público más amplio.

Las plataformas de desarrollo de sistemas multiagente coinciden en algunas características básicas, pero sin embargo, poseen capacidades diferentes y están orientadas a diferentes escenarios. A pesar de que el uso de una plataforma aceleraría el desarrollo de la aplicación, el aprendizaje y uso de estas plataformas no es trivial, y requerirá de un tiempo extra considerable que se sumará al del desarrollo de la aplicación. Por lo tanto, elegir la plataforma adecuada resulta fundamental. Sin embargo, esto es motivo nuevamente de la consulta a expertos que posean los elementos de juicio necesarios para la evaluación de la plataforma adecuada para el escenario de la aplicación.

El objetivo de este trabajo es extender la linea de investigación presentada en [18], evaluando tres plataformas para el desarrollo de sistemas multiagente con el propósito de definir, de manera general, cuando conviene utilizar cada una de ellas. De esta manera, se brindarán algunos elementos de juicio que se pueden tener en cuenta al momento elegir la mejor plataforma para desarrollar una aplicación. Para esto, se han elegido tres plataformas que se encuentran disponibles en internet y que a nuestro entender resultan como las más prometedoras. Lamentablemente, no se ha encontrado en la literatura ninguna especificación de qué criterios deben utilizarse para comparar este tipo de plataformas. Por lo tanto, se proponen algunos criterios que a nuestro entender permiten realizar una comparación general, y que podrían utilizarse para extender la comparación a otras plataformas (ver Sección 5).

Los criterios elegidos para realizar la comparación son: si la plataforma soporta algún lenguaje de comunicación entre agentes (ACL), ya sea FIPA ACL [11] y/o KQML [10], 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 disponibles en Internet, cuán dificultosa es la instalación de dichos paquetes, que tan completa es la documentación y el tipo de interface.

Las plataformas consideradas son: JACK [12] la cual provee un entorno de desarrollo orientado a agentes construido sobre Java y completamente integrado con este lenguaje de programación. MADKit (Multi-agent Development Kit) [17, 9] que ofrece una plataforma multiagente para desarrollar y ejecutar aplicaciones basadas en un paradigma orientado a la organización. Y finalmente el proyecto ZEUS [25, 6] cuyo objetivo es proveer una herramienta de propósito general y personalizable, que pueda ser usada por ingenieros de software con poca experiencia para crear sistemas multiagente. Las tres plataformas comparten la característica de estar implementadas en Java, y puede asegurarse su funcionamiento en Java 1.3.

Además de los criterios antes mencionados y a fin de poder realizar una evaluación de las plataformas en cuanto a su uso, se desarrollo un ejemplo común. Este ejemplo tiene como objetivo principal permitir la consideración de características del uso de las plataformas.

2. JACK

JACK [12, 3] 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, y además incluye las siguientes extensiones específicas para implementar el comportamiento de los agentes: (a) define nuevas clases base, interfaces y métodos (b) provee extensiones a la sintaxis de Java para soportar clases orientadas a agentes y (c) provee extensiones semánticas para soportar la ejecución del modelo.

Incluye a *JACK Agent*, un lenguaje orientado a agentes y utilizado para implementar sistemas de software orientados a agentes. JACK Agent no sólo extiende la funcionalidad de Java, sino que además provee un entorno para soportar un nuevo paradigma de programación. Como JACK fue desarrollado para proveer una extensión de Java orientada a agentes, el código JACK es primero compilado a código Java regular antes de ser ejecutado. La relación entre JACK y Java es análoga a la relación entre los lenguajes C++ y C. 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.

Según se establece en [24], las metas más importantes en el diseño de JACK fueron proveer a los desarrolladores de un producto robusto, estable y "liviano", satisfacer una variedad de necesidades practicas, facilitar la transferencia de tecnología de la investigación a la industria, y permitir la investigación aplicada.

Los agentes JACK no están ligados a ningún lenguaje de comunicación entre agentes particular. Sin embargo, nada previene la adopción de un protocolo simbólico de alto nivel como KQML [10] o FIPA ACL [11], posiblemente integrando software existente en el dominio público. Además, JACK provee una infraestructura de comunicaciones para situaciones donde se requiere alta performance.

Desde la óptica de un programador, JACK ofrece tres extensiones a Java: un lenguaje de agentes, un compilador de agentes y un kernel de agentes. A continuación se describe cada una de ellas.

El lenguaje de agentes JACK: este lenguaje es un superconjunto de Java utilizado para describir un sistema orientado a agentes. Las diferencias con el lenguaje Java son un conjunto de agregados que se detallan a continuación:

- Un pequeño conjunto de palabras claves para la identificación de los componentes principales de un agente.
- Un conjunto de sentencias para la declaración de atributos y otras características de los componentes. Todos los tributos son fuertemente tipados.
- Un conjunto de sentencias para la definición de relaciones estáticas.
- Un conjunto de sentencias para la manipulación de los estados de un agente. Además, el programador puede usar sentencias Java dentro de las componentes de un agente.

Para la conveniencia de los programadores, en particular aquellos con conocimientos en inteligencia artificial, JACK también soporta variables lógicas y sin instanciar. Su semántica es un intermedio entre los lenguajes de programación en lógica (con el agregado del chequeo de tipos del estilo de Java) y SQL embebido.

El compilador de agentes JACK: La segunda extensión es un compilador que convierte las extensiones descriptas anteriormente a clases Java y sentencias que pueden ser cargadas con, y ser llamadas por, otro código Java. El compilador también transforma parcialmente el código de los planes para obtener la semántica correcta de la arquitectura BDI [21, 20].

El kernel de agentes JACK: Finalmente, un conjunto de clases (llamado ***kernel***) provee el soporte de tiempo de ejecución para el código generado. Este incluye:

- La administración automática de la concurrencia.
- El comportamiento normal del agente en la reacción a los eventos y falla de acciones y tareas.
- Una infraestructura "liviana" de comunicaciones de alta performance para las aplicaciones multiagente.

Programación orientada a equipos en JACK

JACK provee una extensión para soportar la programación orientada a equipos (Team Oriented Programming) [5], llamada *SimpleTeam* [24]. La programación orientada a equipos es una variante de la programación orientada a agentes donde la colaboración entre agentes es especificada desde el punto de vista abstracto del grupo como tal. Dentro de la investigación en Inteligencia Artificial, el trabajo en equipos ha sido estudiado desde los principios de 1990, y es un campo de rápido crecimiento.

Diferentes teorías y tipos de equipos han sido propuestos en la literatura. Estas teorías han sido propuestas de acuerdo con las creencias y metas mutuas, donde cada miembro de un equipo intenta alcanzar lo que cree que es una meta del equipo completo. El concepto detrás de este enfoque es que el comportamiento coordinado está especificado o programado desde una perspectiva de alto nivel, y que la maquinaria subyacente mapea luego tales especificaciones en actividades individuales de los agentes involucrados.

La extensión *SimpleTeam* es neutral a la naturaleza de la estructura de un equipo. La única imposición, es que se debe poder clasificar a los miembros de un equipo en término de roles abstractos. La meta de *SimpleTeam* es proveer una infraestructura de software para la especificación del comportamiento coordinado, el cual puede ser posteriormente utilizado para realizar estudios aplicados de organización social.

Aplicaciones en JACK

La mayoría de las aplicaciones desarrolladas en JACK que se encuentran citadas en la literatura han sido construidas por el departamento de defensa DSTO (The Defence Science and Technology Organization) en Australia. Cross y Rönnquist en [7] describen un simulador de combate aéreo, construido con el objetivo de permitir comparaciones con SWARMM [23]. También se está considerando la simulación de operaciones marítimas, así como las tácticas de defensa aérea. Un sistema multiagente para planificación de recursos en operaciones de vigilancia, conocido como *Collection Plan Management System*, ha sido recientemente desarrollado para la división de operaciones de tierra de DSTO, construido como una componente funcional dentro de un entorno CORBA.

3. MADKit (Multi-Agent Development Kit)

MADKit [17, 9] es una plataforma multiagente para desarrollar y ejecutar aplicaciones basadas en un paradigma orientado a la organización. Este paradigma utiliza a los *grupos* y los *roles* como base para construir aplicaciones complejas. MADKit no está asociado a ninguna arquitectura de agentes en particular, permitiendo a los usuarios de esta plataforma implementar libremente sus propias arquitecturas.

MADKit permite el desarrollo de aplicaciones distribuidas de manera muy simple. Para los programadores, consideraciones acerca de componentes distribuidos básicos (como "sockets" o "ports") son totalmente transparentes. Una aplicación desarrollada en MADKit puede ser ejecutada en forma distribuida sin cambiar ninguna línea de código. Los mecanismos de distribución de MADKit no utilizan las técnicas de RMI o CORBA de acceso remoto, lo cual brinda un modo más eficiente de comunicación [17].

La plataforma MADKit está construida alrededor del concepto de *Agente/Grupo/Rol* desarrollado en el contexto del proyecto AALAADIN. MADKit implementa y usa su propio modelo de administración. A continuación se presentan algunas características generales de este modelo, mayores detalles pueden encontrarse en [9].

Un **agente** es especificado como una entidad activa que se comunica, la cual posee roles dentro de los grupos. Esta definición de agentes es intencionalmente general para permitir a los diseñadores de agentes adoptar el modelo de agente más preciso relacionado con sus aplicaciones. El modelo no restringe la arquitectura interna de los agentes. El diseñador es el responsable de elegir el modelo más apropiado de agente de acuerdo a sus necesidades o preferencias.

Los **grupos** son definidos como conjuntos atómicos de agregaciones de agentes. Cada agente es parte de uno ó más grupos. En su forma básica, el grupo es sólo una manera de nombrar un conjunto de agentes. En una forma mas desarrollada, en conjunción con la definición de rol, puede representar cualquier sistema multiagente. Un agente puede ser miembro de n grupos al mismo tiempo. Un grupo puede ser fundado por cualquier agente.

El **rol** es una representación abstracta de la función de un agente, servicio o identificación dentro de un grupo. Cada agente puede manejar múltiples roles y cada rol manejado por un agente es local a un grupo. Manejar un rol en un grupo debe ser requerido por el agente candidato, y no es necesariamente otorgado. Los esquemas abstractos de comunicación son definidos como roles.

El modelo no es una descripción estática de la organización de un agente. También permite definir reglas para especificar la parte dinámica de la organización de un agente.

Principios de diseño

Además de los tres conceptos claves sobre los que está construida la plataforma, esta agrega tres principios de diseño.

- Microkernel: El microkernel es un pequeño y optimizado kernel de agente. Este solo se encarga de las siguientes tareas: control de grupos locales y roles, administración de los ciclos de vida de los agentes, y pasaje local de mensajes.

- Agentificación de servicios: En contraste con otras arquitecturas, MADKit utiliza agentes para implementar servicios como: pasaje distribuido de mensajes, control de migración, etc. Estos servicios son representados en la plataforma como roles en grupos específicos y definidos en una estructura organizacional abstracta. Esto permite un alto grado de personalización, ya que al ser agentes pueden ser intercambiados sin mucho trabajo.
- Modelo de componentes gráfico: El modelo gráfico está basado en componentes gráficas independientes, usando la especificación *Java Beans* en su versión estándar.

Comunidades

Una *"comunidad"* es simplemente un grupo de kernels MADKit conectados. Cuando se diseña una aplicación en MADKit, se pueden crear grupos que pertenezcan a comunidades específicas. Una comunidad, desde el punto de vista de un programador puede ser vista como un espacio para ordenar varios grupos que son usados en la misma aplicación. Este concepto permite particionar una red MADKit logrando: mayor performance, mejor distribución de los recursos y servicios y proveer un entorno seguro para aplicaciones distribuidas.

Aplicaciones en MADKit

MADKit ha sido utilizado en varios equipos de investigación por cerca de dos años, en proyectos que cubren un amplio rango de aplicaciones. Desde la simulación de arquitecturas híbridas para el control de robots submarinos, a la evaluación de redes sociales o el estudio de control multiagente en una línea de producción.

Por ejemplo, WEX, desarrollado por Euriware S.A. [8], es una compleja aplicación MADKit para administración de conocimiento. Unifica la información de diferentes fuentes de datos (bases de datos, herramientas de soporte, motores de búsqueda en la web, etc.) y presenta vistas unificadas de estas fuentes de conocimiento altamente heterogéneas. Los usuarios pueden mantener ontologías compartidas en sus dominios. Los agentes han sido implementados para encapsular los mecanismos para recuperar y transformar la información. Siguiendo una estructura organizacional abstracta, los agentes pueden ser conectados de diferentes maneras para adaptar la plataforma a las necesidades específicas del cliente.

4. ZEUS

ZEUS [25, 6] es una herramienta para construir aplicaciones multiagente colaborativas. que provee un entorno integrado para el desarrollo rápido de sistemas. ZEUS define una metodología de diseño de sistemas multiagente y lo soporta mediante un entorno visual para capturar las especificaciones de los agentes. Estas especificaciones son luego utilizadas para generar el código fuente en Java.

El objetivo del proyecto ZEUS 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.

La herramienta ZEUS consiste de un conjunto de componentes, escritas en el lenguaje de programación Java, que puede ser categorizada en tres grupos funcionales o librerías: una librería de componentes de agentes, una herramienta de construcción de agentes y un conjunto de agentes utilitarios entre los cuales podemos encontrar servidores de nombres, facilitadores y agentes visualizadores. A continuación se describe cada una de ellas.

Librería de componentes de agentes: es una colección de clases que forman los bloques de construcción de los agentes individuales. El contenido de esta librería muestra los puntos identificados antes: entre ellos la comunicación, ontologías, coordinación, etc.

Para la comunicación, la librería de componentes de agentes provee un lenguaje de comunicación entre agentes basado en actos del habla y en performatives, un sistema de pasaje de mensajes asincrónico basado en sockets, un editor para describir ontologías de dominio específico y un lenguaje de representación de conocimiento basado en frames para representar los conceptos del dominio. Para el razonamiento y la coordinación de múltiples agentes, la librería de componentes de agentes provee:

- Un planificador de propósito general y un sistema de planificación preparado para dominios típicos de aplicaciones orientadas a tareas, y un mecanismo cooperativo de resolución de problemas de estas aplicaciones.
- Un motor de coordinación que controla el comportamiento social de un agente.

Herramienta de construcción de agentes: esta herramienta está diseñada para proveer un desarrollo rápido de agentes a alto nivel, ocultando la complejidad de la librería de componentes de agentes. Está conformada por un conjunto de editores diseñados para permitir a los usuarios interactivamente crear agentes mediante una especificación visual de sus atributos. El conjunto actual de editores incluye:

- Un editor de Ontologías para definir los items de la ontología en un dominio.
- Un editor de hechos/variables para describir instancias específicas de hechos y variables, utilizando los templates creados usando el editor de ontologías.
- Un editor de definiciones de agentes para describir los agentes lógicamente. Esto involucra la especificación de las tareas de cada agente, sus recursos iniciales y las dimensiones de su plan.
- Un editor de descripción de tareas para especificar los atributos de las tareas y para resúmenes gráficos de las tareas.
- Un editor de organización para definir las relaciones organizacionales entre los agentes, y las creencias de los agentes acerca de las habilidades de otros agentes.
- Un editor de coordinación para seleccionar el conjunto de protocolos de coordinación con los cuales cada agente estará equipado.

Agentes utilitarios ZEUS: consiste de un servidor de nombres, un facilitador para el descubrimiento de información, y un agente para visualizar o realizar un debugging de sociedades de agentes. Una sociedad de agentes en ZEUS puede contener cualquier número de agentes utilitarios, con al menos un servidor de nombres. Todos los agentes utilitarios son construidos utilizando las componentes básicas de la librería de componentes de agentes, y son en realidad simplificaciones del agente genérico ZEUS.

5. Comparación de las plataformas

El objetivo de esta sección es realizar un análisis comparativo de las tres plataformas consideradas en este trabajo. Brindando, de esta manera, algunos elementos de juicio que pueden tenerse en cuenta al momento elegir una plataforma para desarrollar una aplicación.

Lamentablemente, no se ha encontrado en la literatura ninguna especificación de qué criterios deben utilizarse para comparar este tipo de plataformas. Por lo tanto, se proponen algunos criterios que a nuestro entender permiten realizar una comparación general, y que podrían utilizarse para extender la comparación a otras plataformas (ver Figura 1). Los criterios propuestos son los siguientes:

- Si la plataforma permite construir agentes que puedan utilizar algún lenguaje de comunicación entre agentes estándar cómo KQML o FIPA ACL.
- Si se utiliza alguna arquitectura base para los agentes como BDI por ejemplo.
- Qué tipo de agentes se generan: colaborativos, deliberativos, de negociación, etc.
- Qué lenguajes de implementación de agentes que pueden utilizarse.
- Si permite implementar agentes con código móvil.
- Qué disponibilidad y tipo de licencia tiene.
- Formato de la interface.
- Qué tipo de dificultades pueden encontrarse en la instalación.
- Qué soporte de documentación y ayuda al programador posee.

En la Figura 1 pueden verse los resultados de las evaluaciones.

	JACK	MADKit	ZEUS
ACL soportado		KQML	KQML
Arquitectura base	BDI	Agente/grupo/rol	BDI[1]
Tipo de agentes soportados	Cualquiera	Cualquiera	Deliberativos Colaborativos
Lenguajes soportados para implementación de agentes	Jack	Java [15] Jess [16] Python [19] Scheme [22] BeanShell [1]	Java [15]
Movilidad de código	No detalla	No detalla	No detalla
Disponibilidad	on-line	on-line	on-line
Licencia	gratis 30 días	GPL/LGPL	Mozilla public
Interface	GUI	GUI	GUI
Instalación	Simple	Simple	Simple
Documentación	Muy completa	Pobre	Pobre
Ayuda	Manual	On-line	Manual

Figura 1: Comparación general de las plataformas

Es importante destacar, que una evaluación exhaustiva debería incluir la implementación de una aplicación multiagente lo suficientemente compleja, como para detectar efectivamente los puntos fuertes o débiles de presente cada plataforma al momento de ser utilizada. Sin embargo, además del costo asociado a esta tarea, resulta difícil definir que significa "lo suficientemente compleja". Como una solución de compromiso, en este trabajo se incluyen las conclusiones obtenidas de la implementación de un problema sencillo en las plataformas evaluadas. El ejemplo seleccionado consiste en un sistema de calefacción/refrigeración central.

El sistema es el encargado de sensar la temperatura del ambiente y de informarla al sistema, así como también de registrar la temperatura deseada por los usuarios. Por otro lado tenemos al termostato, este es el encargado de mantener la temperatura del ambiente, en un cierto rango de aceptación, con la temperatura deseada que es indicada por los usuarios. Tiene, además, el control sobre la caldera y el aire acondicionado con lo que trata de manejar la temperatura del ambiente.

Considerando una arquitectura BDI el deseo del agente termostato es mantener la temperatura actual dentro del rango de aceptación con respecto a la temperatura deseada. Para lograr alcanzar este deseo cuenta con las intenciones de aumentar la temperatura o disminuir la temperatura. Las acciones que el agente puede realizar son: encender la caldera, apagar la caldera, encender el aire, o apagar el aire. Para mantener la temperatura del ambiente, envía mensajes a la caldera y al aire acondicionado para encenderlos o apagarlos según sea necesario. Una vez encendidos, el termostato solicita al sistema la temperatura actual y de esta manera decidir si ya se alcanzó la temperatura deseada y pasar a un estado *idle*.

En cuanto a la implementación del ejemplo propuesto podemos decir que las plataformas propuestas poseen las siguientes características:

- JACK: Para un uso eficiente de esta plataforma, es necesario tener en mente el diseño del sistema y de los agentes que lo componen ya que posee una separación de las componentes de un agente, separa en: capacidades, planes y eventos (entrantes y salientes) de los agentes. Para la implementación del ejemplo propuesto, fue necesario como primer paso la definición de las capacidades de cada uno de los agentes, posteriormente la especificación de los mensajes que utilizan los agentes. Como tercer paso se incluyeron los agentes que componen el sistema, asociándole a cada uno los mensajes y capacidades correspondientes. Finalmente, se detallaron los planes asociados a cada agente, relacionando estos planes con los mensajes que los inician, lo que se debe realizar en el plan y los mensajes que se contestan para indicar que el plan fue ejecutado, como por ejemplo subir la temperatura actual mediante el encendido de la caldera si la temperatura actual es menor a la temperatura deseada; y finalmente se definen los agentes que componen el sistema y se les asocian los mensajes que estos deben *escuchar* y los planes a los que deben reaccionar. Como último paso se creo un archivo que realice la creación de todos los agentes y la inicialización del sistema.

- MADKit: Para esta plataforma los agentes están caracterizados por roles y grupos. Como pasos necesarios para la implementación en esta plataforma dotamos a cada agente con una capacidad determinada, esta capacidad determina el rol que el agente cumple dentro del grupo en el que se desenvuelve. Por ejemplo: el termostato posee el rol de mantener la temperatura deseada en relación a la actual; el sistema tiene el rol de informar la temperatura actual y monitorear la deseada por el usuario. Una vez que el sistema comienza

su ejecución, los agentes negocian los roles de acuerdo a sus capacidades dentro del grupo y de ser aceptado, el agente será el encargado de desempeñar el rol.

- ZEUS: En esta plataforma, el desarrollo se basa en las tareas que realizan los agentes. Como primer paso se definen las ontologías que se utilizarán en el sistema. Una vez que las ontologías están definidas, se especifican las tareas que realizan los distintos agentes pero sin detallar que tarea corresponde a que agente. Para el ejemplo, se le define al termostato la tarea de mantener la temperatura actual en relación a la deseada por el usuario y al sistema la tarea de informar al termostato la temperatura actual. Posteriormente se detallan los agentes y se les asocia a cada uno las tareas que estos deben realizar dentro del sistema. Como último paso se realiza la generación de código, la cual incluye la generación de los agentes y de las tareas que estos realizan. Además de incluir un visor de la sociedad de agentes.

6. Otras plataformas

Incluimos a continuación una descripción resumida de otras dos plataformas citadas en la literatura. Lamentablemente, hasta el momento no hemos podido acceder a su implementación, por lo cual no se incluye una evaluación de las mismas.

JADE (Java Agent Development Framework)

JADE [13, 2] es un entorno que simplifica la implementación de sistemas multiagente mediante una capa de soporte (middle-ware) que respeta las especificaciones FIPA [11] y con un conjunto de herramientas para el desarrollo y debugging. La plataforma puede ser distribuida en varias máquinas (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. 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.

JAFMAS (Java Framework for Multi-agent Systems)

JAFMAS [14, 4] 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.

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 [10]). 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.

7. Conclusiones y Trabajo Futuro

En este trabajo se propusieron algunos criterios que permiten realizar una comparación general entre plataformas de desarrollo de sistemas multiagente. Con los criterios propuestos fueron evaluadas tres plataformas que se encuentran disponibles en internet y que a nuestro entender resultan como las más prometedoras. Esta evaluación permitió obtener elementos de juicio que puedan tenerse en cuenta al momento elegir una plataforma para desarrollar aplicaciones.

La Figura 1 muestra los resultados obtenidos en la evaluación, donde puede observarse que aunque existen elementos comunes, cada plataforma propone un enfoque diferente para el diseño de una aplicación. JACK es una plataforma que provee su propio lenguaje de programación de agentes. Esta trabaja con la estructura de agentes BDI y en particular con la ejecución de planes. MADKit es una plataforma basada en el concepto agente/grupo/rol que permite el trabajo con agentes de cualquier tipo. Por su parte, la plataforma ZEUS está basada en la arquitectura BDI y permite el desarrollo de agentes en lenguaje Java los cuales se centran en ontologías y tareas.

Como trabajo futuro se planea realizar una evaluación más profunda de estas plataformas mediante el desarrollo de aplicaciones más complejas. Además se planea considerar nuevas plataformas para su evaluación y comparación.

Referencias

[1] BEANSHELL. *Lenguaje BeanShell*. 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] Hodgson Busetta, Rönnquist and Lucas. Jack intelligent agents - components for intelligent agents in java. Agent Oriented Software Pty. Ltd., 1999.

[4] 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.

[5] Cohen and Levesque. Teamwork. Nous, 1991.

[6] Collins and Ndumu. *The Zeus Agent Building Toolkit*. ZEUS Technical Manual, 1999.

[7] Cross and Rönnquist. A java agent environment for simulation and modelling. In *Proceedings of the Fourth International SimTect Conference*, 1999.

[8] Groupe Euriware. *Wex Cooperative KM solution using Java technology*. http://wwws.sun.com/software/java/javacenters/locations/francegroupe.html, 2002.

[9] Ferber and 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.

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

[11] FIPA. *Foundation for Intelligent Physical Agents*, 2002.

[12] JACK. *JACK Intelligent Agents*. Agent Oriented Software Pty. Ltd., http://www.agent-software.com/, 2003.

[13] JADE. *Java Agent Development Framwork*. TILAB, http://www.sharon.cselt.it/projects/jade/, 2002.

[14] JAFMAS. *Java Based Framework for Multi Agent Systems*. University of Cincinnati, http://www.ececs.uc.edu/ abaker/JAFMAS, 2002.

[15] JAVA. *Lenguaje Java*. http://java.sun.com, 2003.

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

[17] MADKIT. *Multi-Agent Development KIT*. http://www.madkit.org, 2002.

[18] Tulio Marchetti and Alejandro J. García. Plataformas para desarrollo de sistemas multi-agente. un análisis comparativo. *Actas del V Workshop de Investigadores en Ciencias de la Computación. Tandil, Argentina*, pages 389–393, 2003.

[19] PYTHON. *Lenguaje de script Phyton*. JYTHON, http://www.jython.org/, 2003.

[20] Rao and Georgeff. Modeling rational agents within a bdi-architecture. In *Proceedings of Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, 1991.

[21] Rao and Georgeff. Bdi agents: from theory to practice. In *Proceedings of the 1st. International Conference on Multi-Agent Systems*. ICMAS-95, 1995.

[22] SCHEME. *Lenguaje de script Scheme*. KAWA, http://www.gnu.org/software/kawa/, 2003.

[23] Selvestrel Tidhar and Heinze. Modelling teams and team tactics in whole air mission modelling. In *Proceedings of the Eighth International Conference on Industrial Engineering Applications of Artificial Intelligence Expert Systems*. Gordon and Breach Publishers, 1995.

[24] Padgham Winicoff and Harland. Simplifying the development of intelligent agents. RMIT University, Melbourne, 2001.

[25] ZEUS. *The ZEUS Agent Building Tool*. British Telecommunications, http://more.btexact.com/projects/agents/zeus/, 2002.