Implementación de transacciones distribuidas usando agentes inteligentes

Implementation of distributed transactions using intelligent agents

Valentín Álvarez Hilario[*]
René E. Cuevas Valencia[**]
José MarioMartínez Castro[***]

Fecha de recepción: 15 de abril del 2011
Fecha de aceptación: 16 de junio del 2011

Resumen

Los desarrollos tecnológicos en nuestros días, requieren herramientas de software y modelos acordes con ellos, por tal razón, el desarrollo de aplicaciones basadas en agentes inteligentes con el uso de lenguajes de programación como es Jade, que aplica estándares internacionales como Fipa, redunda en una mayor facilidad para la administración y el desarrollo de transacciones distribuidas.

Por esta razón, el uso de las bases de datos distribuidas en desarrollos orientados a los agentes resuelve, en gran medida, las problemáticas en desarrollos distribuidos como es la confiabilidad, la tolerancia a fallos y la seguridad. Jade cuenta con un administrador de transacciones distribuidas que permite que las aplicaciones de sistemas multiagentes solo se programen en la solución específica del problema por resolver, dejando de lado los problemas inherentes a conectividad entre equipos de cómputo.

Palabras clave: agentes inteligentes, transacciones distribuidas, Fipa, Jade.

[*] Universidad Autónoma de Guerrero. Unidad Académica de Ingeniería, Chilpancingo Guerrero; México. Correo electrónico: valentin_ah@yahoo.com

[**] Universidad Autónoma de Guerrero. Unidad Académica de Ingeniería, Chilpancingo Guerrero; México. Correo electrónico: reneecuevas@hotmail.com

[***] Universidad Autónoma de Guerrero. Unidad Académica de Ingeniería, Chilpancingo Guerrero; México. Correo electrónico: jmmtzc@hotmail.com.

Introducción

Es correcto puntualizar que la evolución de la computación ha dado origen a diversas formas de organizar la información, debido a que la mayoría de las bases de datos son diseñadas e implementadas respondiendo a diferentes necesidades y circunstancias, así como con distintos recursos, lo cual da como resultado una amplia heterogeneidad en los diversos depósitos de información. Esta última es el recurso clave tanto en iniciativa privada, gobierno o ambiente académico. La forma como la información se encuentra organizada es diversa, múltiples métodos de acceso y paradigmas para el manejo de información son utilizados.

La necesidad de manipular información en su conjunto, requiere acceder a la información desde distintas fuentes de datos y no es razonable tener que aprender distintos métodos de manipulación de datos y aun es menos razonable pensar que los poseedores de datos tengan que trasladar toda su información a un modelo de datos común y con una sola forma de manipularlos.

Lo anteriormente expresado nos da un panorama de la problemática más generalizada en nuestros días, que ha sido la diseminación de las bases de datos, de acuerdo a las necesidades específicas de cada usuario final o de cada área en particular; sin menoscabar la importancia que para el personal directivo, sea clave y trascendental, la integración de esta información para la toma de decisiones. Por lo anterior, es necesario disponer de sistemas de bases de datos que puedan operar sobre una red distribuida y puedan compaginar una mezcla heterogénea de computadoras, sistemas operativos, enlace de comunicaciones y sistemas manejadores de base de datos locales.

Muchas empresas o instituciones para el registro de sus operaciones cuentan con diferentes tipos de computadoras y diferentes sistemas manejadores de bases de datos. En la mayoría de los casos, este ambiente debe ser preservado, pero también existe la necesidad de compartir información en una o más bases de datos globales. Se requiere la manipulación de datos global para recuperar y actualizar información semánticamente similar en diferentes nodos y con diferentes representaciones de datos.

Bases de datos distribuidas

Las bases de datos distribuidas eliminan muchas de las deficiencias de bases de datos centralizadas y cabe más naturalmente en las estructuras descentralizadas de muchas organizaciones.Una definición de una base de datos distribuida es una colección de datos que son distribuidos sobre diferentes computadoras de una red de estas (Pelagatti, Politecnico di Milano y Ceri, 1985).

Por otra parte, un sistema de administración de base de datos distribuida soporta la creación y el mantenimiento de bases de datos distribuidas. Varios sistemas distribuidos comercialmente disponibles fueron desarrollados por los vendedores de sistemas de administración de base de datos centralizados. Ellos contienen componentes adicionales que extienden la capacidad de Database Management System (DBMS) centralizados para soportar la comunicación y la cooperación entre varias instancias de DBMS que son instalados en diferentes sitios de una red de computadoras. Los componentes de software (Pelagatti, Politecnico di Milano y Ceri, 1985) que son típicamente necesarios para construir bases de datos en este caso son:

1) El componente de administración de base de datos (DB).
2) El componente de comunicación (DC).

3) El diccionario de datos (DD) que extiende la información presente acerca de la distribución de los datos en la red.
4) Los componentes de base de datos distribuida (DDB).

Administración de transacciones

Con la introducción del concepto de transacción, se expande el papel del monitor de ejecución distribuida. Este consiste de dos módulos: el administrador de transacciones (TM) y el despachador (SC). El administrador de transacciones es responsable de coordinar la ejecución en la base de datos de las operaciones que realiza una aplicación. Por otra parte, el despachador es responsable de implementar un algoritmo específico de control de concurrencia para sincronizar los accesos a la base de datos.

Un tercer componente que participa en el manejo de transacciones distribuidas es el administrador de recuperación local, cuya función es implementar procedimientos locales que le permitan a una base de datos local recuperarse a un estado consistente después de una falla.

La programación orientada a agentes

Las técnicas orientadas a agentes representan una forma de diseñar sistemas de software complejos. Estos sistemas, inherentemente distribuidos, surgen por el aumento de las redes de computadoras y de la información disponible; también para hacer frente al desafío de interconectar arquitecturas heterogéneas. Los agentes de software y los sistemas multiagente representan una nueva forma de analizar, diseñar e implementar estos sistemas de software de aplicaciones distribuidas.

La tecnología de agentes ha mantenido un alto interés comercial y un potencial tecnológico importante para solucionar problemas actuales de los sistemas de computadoras. Entre estos problemas se encuentran: facilitar la interfaz al usuario, reducir el problema del ancho de banda en las redes, solucionar los inconvenientes por el uso de redes no seguras, como también hacer posible la vieja promesa de diseñar sistemas de computadora que sean adaptables, autónomos y que aprendan de sus experiencias, es decir, que exhiban algún grado de inteligencia, aunado a todo ello, está el hecho de interconectar bases de datos heterogéneas manejadas como un todo (bases de datos distribuidas).

Definición de agentes

El término agente ha sido usado desde hace tiempo en inteligencia artificial y en computación distribuida. Sin embargo, aún es complicado encontrar una única definición que incluya todas las características que los investigadores toman en consideración al hablar de un agente y excluya todas las otras que no tiene que tener. Sin embargo, a raíz de la creación del organismo internacional llamado "Fundación de Agentes Físicos Inteligentes" (FIPA, por sus siglas en inglés) se ha propuesto la siguiente definición, la cual se ha tomado como estándar: "Es un proceso computacional que implementa la autonomía y la comunicación funcional de una aplicación" (FIPA, 2002).

Breve historia

En 1977, Carl Hewittis propuso el concepto de un objeto autocontenido, interactivo y de ejecución concurrente, llamado actor. Este objeto tenía algún estado interno encapsulado y podría responder a mensajes de otros objetos similares. "Es un agente computacional, que tiene una dirección de correo elec-

trónico y un comportamiento. Los actores se comunican por paso de mensajes y desarrollan sus acciones concurrentemente" (Hewitt, 1977, p. 131).

Podemos dividir la investigación en agentes en dos ramas principales (Gómez y Zea, 2011), la primera en el periodo comprendido entre 1977 hasta 1990 y la segunda desde 1990 hasta nuestros días. En la primera, las investigaciones se concentraron principalmente en tipos de agentes repartidores, con modelos simbólicos internos, también en aspectos de discusión macros como teoría de arquitectura, teoría de lenguaje, la interacción y la comunicación entre agentes, la descomposición y la distribución de tareas, coordinación y cooperación, resolución de conflictos vía negociación, etc. su objetivo fue especificar, el análisis y el diseño de sistemas integrados incluyendo múltiples agentes colaborativos. Estos puntos de agentes, enfatizaban en las sociedades de agentes más que en agentes individuales. No obstante, a principio de la década de los noventa, los tipos de agentes que se siguen investigando y desarrollados es ahora mucho más amplio e incluyen una preocupación por la integración de bases de datos y sistemas de inferencia.

Características de los agentes

- Reactivos: los agentes son capaces de detectar cambios en su ambiente de ejecución y actuar con base en estos.
- Autónomos: se refiere al hecho de tener control sobre sus propias acciones, es decir, ellos deciden cuándo, cómo y qué hacer ante la situación que se les presente.
- Comportamiento colaborativo: puede trabajar colectivamente para lograr una meta en común.
- Habilidad de comunicación: la habilidad para comunicar con personas y otros agentes con lenguaje más parecido al humano "expresión de actos" con protocolos típicos símbolo-nivel programa-a-programa.
- Capacidad inferencial: puede actuar en la especificación de tareas abstractas usando antes el conocimiento de metas generales y métodos preferidos para lograr flexibilidad; metas más allá de la información dada, y puede tener modelos explícitos de sí mismos, usuario, situaciones u otros agentes.
- Continuidad temporal: persistente de identidad y estado sobre periodos largos.
- Delegación de tareas: se relaciona con el hecho de que el usuario puede desentenderse de una tarea en específico y delegarla para que otro ente la realice. En este caso, el usuario delega ciertos trabajos al agente, indicándole por dónde empezar a trabajar y después solo solicita los resultados.
- Personalizables: ofrecen la posibilidad de personalizar sus atributos.
- Adaptables: ser capaz de aprender y mejorar con la experiencia.
- Móviles: ser capaz de migrar de una manera propia, dirigida lejos de su plataforma a otra.
- Son paralelizables: los agentes móviles pueden crear una serie de clones en la red, por lo que un uso potencial de esta tecnología es administrar tareas en forma paralela.

Usualmente, los agentes son desarrollados como parte de un sistema multiagente (SMA). La tendencia a la interconexión total entre computadoras hace evolucionar las arquitecturas computacionales hacia un conjunto de agentes que representan a usuarios, servicios o datos [5]. Un paradigma usado consiste en la publicación de los servicios por parte del agente de recursos; mientras que el agente de usuario usa los servicios para encontrar lo que necesita y realizar consultas, en consecuencia.

Características de los sistemas multiagente

Cada agente tiene información incompleta y no posee todas las capacidades para resolver el problema por sí mismo, así cada agente tiene un punto de vista limitado.

- No existe un sistema de control global.
- Los datos son descentralizados.
- La computación es asincrónica.

Campo de acción

Los agentes se usan desde algunos años atrás, en una gran variedad de aplicaciones, que van desde pequeños asistentes personales, pasando por aplicaciones para la búsqueda y la recuperación de información de bibliotecas digitales hasta complejos sistemas de control de tráfico aéreo. Entre los que se encuentran:

Asistentes personales. Emplean técnicas de aprendizaje para formar un modelo con las preferencias del usuario y así ayudar en la organización y realización de tareas. Estos asistentes son empleados para administración de compromisos, recomendar libros, películas, entretenimiento; por ejemplo CAP (Mitchell, 1994) aprende las preferencias del usuario sobre sus reuniones(Riecken, 1994), soporta el trabajo cooperativo entre personas y agente que recomienda música, son algunos ejemplos de asistentes personales.

Agentes para filtrado de información. Actualmente, la cantidad de información disponible a diario es enorme (vía diarios electrónicos, correos electrónico, canales de noticias), la idea de estos agentes es brindar al usuario aquellas porciones de información que son significativas para ellos. Máximo (Lashkari, 1994), administrador de correos electrónicos y NEWT (Maes, 1994), agente para filtrar noticias son algunos ejemplos.

Tutores inteligentes: construyen una relación con el usuario a fin de educarlo. Para ello, se emplean agentes con técnicas de aprendizaje y razonamiento que brindan ayuda a las necesidades de un usuario particular. Dicha ayuda es brindada proactivamente, es decir, el sistema detecta una necesidad del usuario brindándosela antes de que éste la requiera. Por ejemplo COACH (Selker, 1994) aprende del usuario y se adapta para brindarle ayuda mientras escribe.

Comercio electrónico: se intenta que algunas de las decisiones involucradas en una transacción sea realizada por agentes. Estos son algunos ejemplos de agentes para comercio electrónico: Kasbah (citado en Chavez, 1996) define un "lugar de comercio" electrónico donde los agentes pueden comprar y vender bienes; asimismo, BargainFinder (citado en Krulwich, 1996) busca por los discos compactos más baratos.

Estas aplicaciones -aparentemente tan diferentes- comparten entidades similares que pueden ser abstraídas en una misma arquitectura de software, es más, estos sistemas fueron desarrollados desde cero y no existía una técnica unificada para su diseño. Hasta que un grupo de personas se unieron en 1997 formando la FIPA (Foundation for Intelligent Phisic Agents) sin fines de lucro con el propósito de estandarizar un modelo para la elaboración de lenguajes, o en su caso, para desarrollar aplicaciones con agentes.

Lenguajes orientados a agentes

La ingeniería de software basada en agentes surge desde la necesidad de contar con herramientas que permitan la operación entre distintos sistemas compuestos de agentes. Esta ingeniería también es usualmente comparada con la programación orientada a objetos (POO); en el sentido de que un agente,

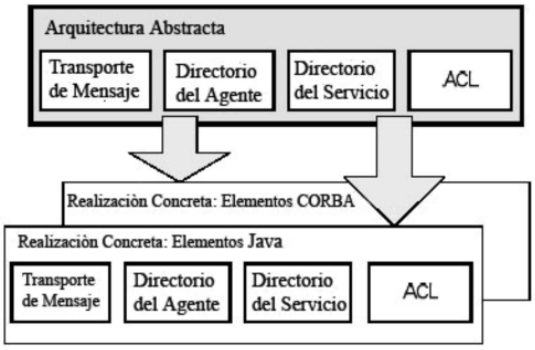
al igual que un objeto, provee una interfaz basada en mensajes hacia sus estructuras de datos internas y algoritmos

Las técnicas orientadas a agentes representan una nueva forma de analizar, diseñar y construir sistemas de software complejos (Jennings, 2000) [12]. La realización de sistemas de software distribuidos y heterogéneos es el desafío que la ingeniería de software debe solucionar, brindando herramientas para tal fin.

Dentro de estos modelos, también tenemos al modelo elaborado por la FIPA, en el cual se no refiere a ningún lenguaje de programación, ningún tipo de comunicación en especial, pero sí toma en cuenta a los más comerciales para su diseño (CORBA, DCOM, LDAP X.500 y MQ series). En lo que respecta a la seguridad, solo toma en consideración las que se pueden implementar mediante el modelo (encriptación y validación de mensajes), aunque hace referencia que la seguridad será implementada de acuerdo con las necesidades de la aplicación.

A continuación, se presenta el modelo abstracto desarrollado por la FIPA y una descripción de este:

Figura 1. Arquitectura abstracta de la FIPA



La arquitectura abstracta de la FIPA define, en un nivel abstracto, cómo dos agentes pueden ser localizados y cómo se pueden comunicar entre ellos, registrándose e intercambiando mensajes. Los agentes se comunican intercambiando mensajes y son codificados en un Lenguaje de Comunicación de Agentes (ACL).

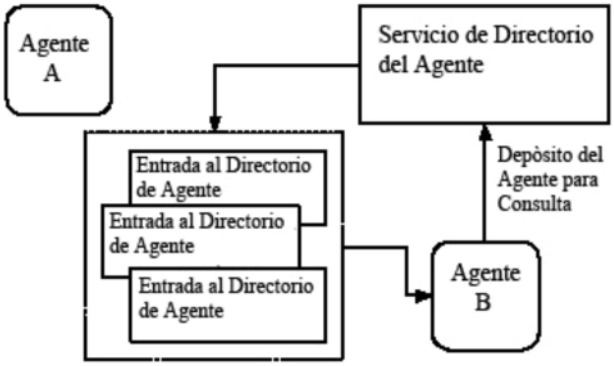
Los servicios proveen servicios de soporte para agentes, además de un número de servicios estándar incluyendo el servicio de directorio de agentes y el servicio de transporte de mensaje. Esta arquitectura abstracta de la FIPA define un modelo de servicio general que incluye un servicio de directorio de servicios.

Al iniciar un agente, debe ser provisto con un servicio principal, típicamente, este servicio será el *servicio de directorio*, que agregará un conjunto de localizadores de servicio para que estén disponibles los servicios de soporte en el ciclo de vida del agente, como servicio de transporte de mensaje, servicio de directorio de agente y servicio de directorio de servicio.

El rol básico del servicio de directorio de agentes es proveer una ubicación donde los agentes registran sus descripciones como las entradas del directorio de agentes. Otros agentes pueden buscar la entrada del directorio de agentes para encontrar con los que ellos desean interactuar. Los agentes pueden localizar el servicio de directorio de agentes para localizar el agente con el cual se comunicarán:

El papel básico del servicio de directorio de servicios, es proveer un medio consistente por el cual los agentes y los servicios pueden descubrir otros servicios. Operacionalmente el servicio de directorio de servicios provee una ubicación en la que los servicios pueden registrar su descripción de servicios como entradas de directorios de servicio. También

Figura 2. Consulta del directorio



los agentes y servicios pueden buscar el servicio de directorio de servicios para localizar servicios apropiado a sus necesidades.

En la FIPA, los agentes se comunican enviando mensajes, tres aspectos fundamentales de la comunicación de mensajes, entre los agentes son: la estructura del mensaje, la representación del mensaje y el transporte del mensaje. La estructura de un mensaje es: clave-valor-tupla y es escrito en un lenguaje de comunicación de agentes como la FIPA ACL.

El contenido del mensaje es expresado en un lenguaje de contenido, como KIF o SL.

En la FIPA, los agentes son previstos para comunicarse con los demás, aquí se hacen algunas consideraciones:

- Cada agente tiene un nombre de agente.
- Este nombre es único y sin posibilidades que se cambie.
- Cada agente tiene uno o más descripciones de transporte que son usados por otros agentes para enviar un transporte de mensaje.
- Cada descripción de transporte corresponde a una forma particular del transporte de mensaje, como IIOP, SMTP, o HTTP.
- Un transporte es un mecanismo para transferir mensajes.
- Un transporte de mensaje es un mensaje que se envía desde un agente a otro en un formato (o codofocado) que es apropiado al transporte que va a ser usado (ver figura 3).

Hay muchos aspectos de seguridad que pueden ser provistos en los sistemas de agentes. Hay una simple forma de seguridad: validación de mensajes y la encriptación. En la validación de mensajes, puede ser enviado de manera que alguna modificación durante la transmisión es identificada. En la encriptación, un mensaje es enviado encriptado de forma que entidades no autorizadas no pueden comprender el contenido del mensaje. Estas características son acomodadas mediante representaciones codificadas y el uso adicional de atributos en la envoltura (ver fig.4).

Lenguaje de programación Jade

Jade (Java Agent Development Framework) es una plataforma desarrollada íntegramente en Java para la creación de sistemas multiagente. Además de proporcionar una API para la creación de agentes y elementos rela-

Figura 3. Descripción de tipo de transporte de mensaje

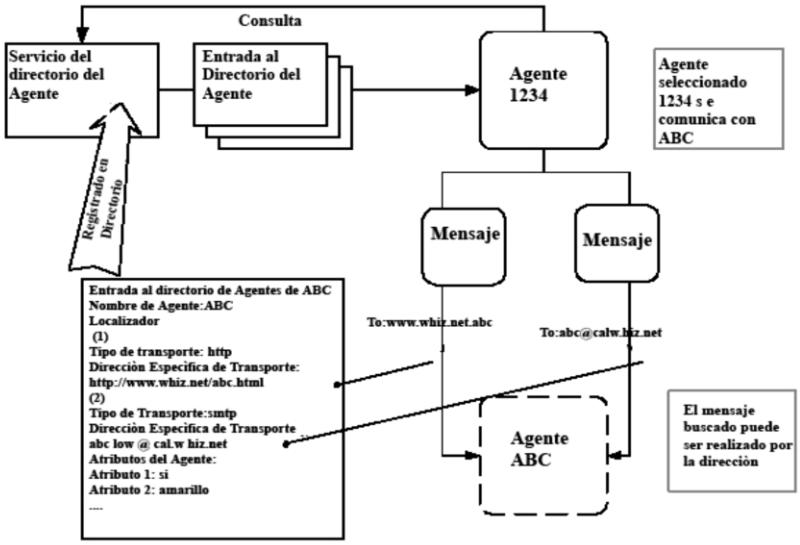
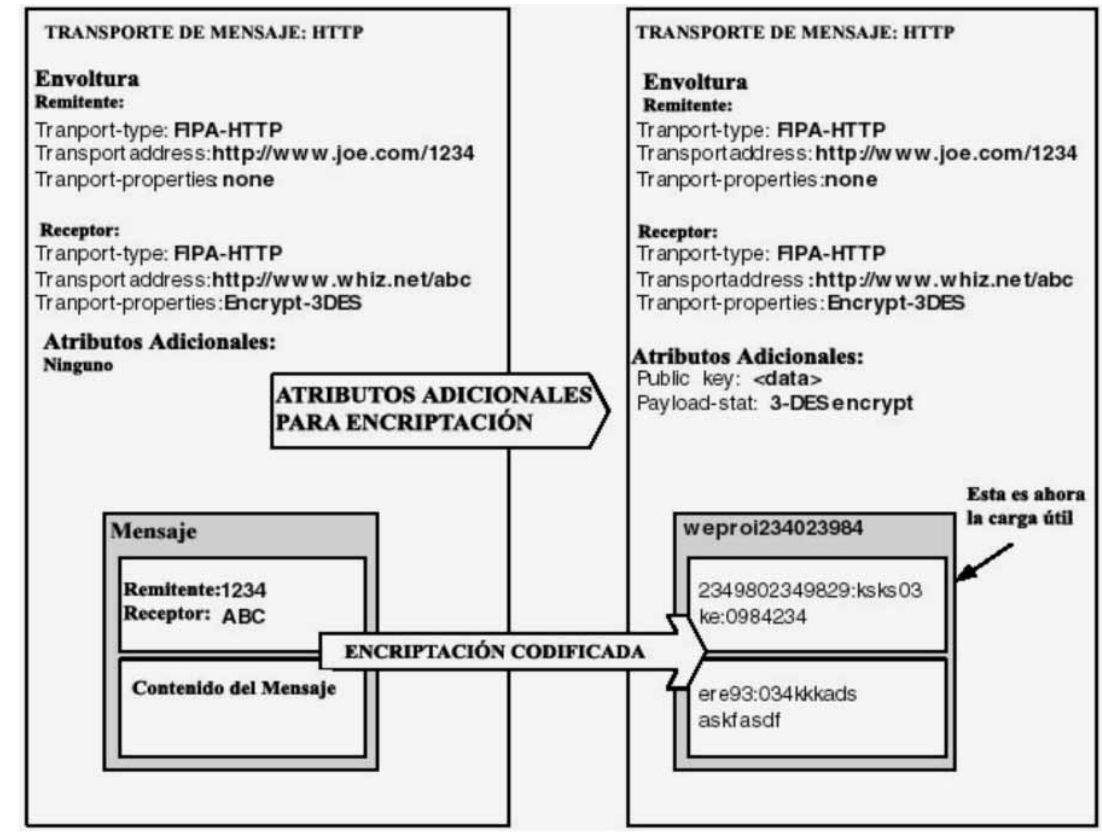


Figura 4. Encriptando un mensaje payload



cionados, presenta una interfaz gráfica y una serie de herramientas para el control y la depuración de sistemas durante el desarrollo de este (Jade, 2004). Además, permite la ejecución de diversos agentes en diferentes plataformas de forma concurrente e, incluso, la posibilidad de desplazar estos agentes de una máquina a otra. Una de las principales características de la plataforma es que los desarrollos se encuentran dentro del estándar de la FIPA. El intercambio de mensajes entre agentes, así como su rendimiento se corresponderán con lo especificado con este estándar.

Jade viene acompañado de una gran cantidad de documentos. El problema es que la información para la creación de un sistema multiagente utilizando las herramientas proporcionadas por la plataforma se encuentra repartida y dispersa entre toda esta documentación. Por último cabe destacar que Jade es software libre y que se distribuye bajo los términos de la licencia LGPL (Lesser General Public License Version 2).

Características del lenguaje

Para iniciar la ejecución de la plataforma, se ejecuta, mediante la máquina virtual de Java, el objeto jade.Boot. Este objeto acepta una serie de parámetros que permitirán indicar los agentes que se desean ejecutar, los contenedores por crear, el host donde ejecutar los agentes, si se muestra la interfaz gráfica asociada al agente RMA, etc.

Para una descripción más detallada de los distintos parámetros con los que se puede llamar a jade.Boot podremos consultar el *Jade's Administrator Guide* que encontraremos entre la documentación presente en la distribución de Jade.

Para que un agente o un conjunto de agentes sean creados al inicializar la plataforma Jade se debe ejecutar jade.Boot de la siguiente forma:

```
java jade.Boot nombre1:clase1 nombre2:clase2 ...
```

Para cada agente deberemos indicar su nombre y la clase a la que instancía -recordemos que cada agente será la instancia de una determinada clase- separado por dos puntos. Así, por ejemplo, al crear un agente de la clase ClaseAgente con el nombre agenteEjemplo, se debe teclear lo siguiente:

```
java jade.Boot agenteEjemplo:ClaseAgente
```

Si, además, quisiéramos que se mostrara la interfaz del agente RMA para poder depurar el sistema multiagente, se debe teclear lo siguiente:

```
java jade.Boot -gui agenteEjemplo:ClaseAgente
```

Como se puede observar, primero se indican las opciones y después la lista de agentes que se van a crear.
La plataforma Jade permite la ejecución de un sistema de agentes en el que éstos puedan estar repartidos entre diversos hosts. Para ejecutar un agente en remoto no tendríamos más que hacer lo siguiente:
java jade.Boot -container -host nombreHost nombre:clase

También es posible crear un RMA que ponga en contacto la plataforma local con la remota. Esto significa que desde el RMA de ambas plataformas podremos contemplar los cambios producidos en el sistema de agentes e interactuar con dichos agentes. Para ello, tenemos que ejecutar:

```
java jade.Boot -container -host nombreHost
RMA1:jade.tools.rma.rma
```

Programar estos sistemas multi-agente implica crear una serie de archivos Java. En concreto, se debe crear, como mínimo, un archivo Java por cada tipo de agente que vaya a estar presente en el sistema, creando una clase derivada de la clase Agent. También es

bastante probable que se creen otros archivos Java para definir la ontología empleada por los agentes del sistema.
Esta plataforma que bien se puede mostrar de manera gráfica o estar oculta representa la parte medular de este documento, ya que con la interfaz gráfica se puede dar cabida a la administración de dichas transacciones de manera sencilla, aunado al complemento del desarrollo de la aplicación específica de agentes.

Ontología

Una ontología consiste en un conjunto de información que es útil en el dominio del problema que se está tratando. Los agentes intercambiarán información según el formato de los hechos especificados en la ontología. Una ontología en Jade es una instancia de la clase jade.content.onto.Ontology, en la cual se definen los esquemas de la estructura de los predicados, acciones de los agentes y conceptos relevantes al dominio del problema. Estos esquemas son instancias de las clases PredicateSchema, AgentActionSchema y ConceptSchema incluidas en el paquete jade.content.schema. Además, para cada uno de estos elementos definidos en la ontología, se deberá crear una clase asociada.

Para cada uno de los esquemas se definirá una serie de campos –por ejemplo, se puede definir el concepto libro con los campos título y autor–. Estos campos se corresponderán con una variable de la clase correspondiente al esquema y, por tanto, tendrán asociado un tipo.

Para explicar mejor el procedimiento, se hará mediante un ejemplo. Supongamos una aplicación en la que un agente deberá comunicar a otro la dirección IP y el número de puerto necesarios para acceder a un determinado recurso. De igual forma, este segundo agente podría enviar un mensaje a un tercer agente en el que le indicara que se debe conectar a esa dirección y puerto. Por tanto, los elementos necesarios para definir la ontología serían:

- Un concepto llamado URL, que contuviera la dirección IP y el número de puerto.
- Un predicado llamado ConexionA, que hiciera referencia a un concepto de tipo URL. Esto es necesario porque en los mensajes entre agentes no se pueden enviar conceptos, solo predicados o acciones.
- Una acción llamada Conecta, que hará referencia a un concepto de tipo URL e indicará al agente que lo reciba, que deberá conectarse a la dirección IP y puerto especificados.

Posteriormente, se debe crear el archivo java que contendrá el objeto de la clase Ontology, que será el encargado de contener la estructura de dichos elementos. De lo anterior, se pueden extraer las siguientes conclusiones:

- Cada esquema añadido a la ontología está asociado a una clase java. Por ejemplo, el esquema para el concepto URL está asociado a la clase URL.java.
- Cada campo de un esquema debe tener un nombre y un tipo. Este tipo se debe corresponder con un determinado esquema. Para los tipos básicos (int, String, etc.) se pueden usar los campos correspondientes de la clase BasicOntology.
- Un campo puede ser declarado *optional* con lo que su valor puede ser null. En otro caso el campo es considerado *mandatory*.
- Otro de los atributos que se le puede dar a un campo (como *optional*) es *unlimited*, con lo que la cardinalidad de ese campo podrá ser mayor que 1.
- Un esquema puede tener superesquemas. Esto se hace mediante el método addSuperSchema.

Cada esquema incluido en la ontología está asociado con una clase Java. La estructura de estas clases debe ser coherente con sus esquemas asociados. Más en concreto, deben obedecer las siguientes reglas:

1. Implementar la interfaz adecuada:
- Si el esquema es un ConceptSchema, la clase debe implementar (directa o indirectamente mediante herencia) la interfaz Concept.
- Si el esquema es un PredicateSchema, la clase debe implementar (directa o indirectamente mediante herencia) la interfaz Predicate.
- Si el esquema es un AgentActionSchema la clase debe implementar (directa o indirectamente mediante herencia) la interfaz AgentAction.

2. Tener las relaciones de herencia adecuadas. Por ejemplo, si S1 es un super-esquema de S2 entonces la clase C2 asociada al esquema S2 debe extender la clase C1 asociada al esquema S1.

3. Tener las variables miembro y los métodos de acceso adecuados:

- Por cada campo en el esquema S1 con nombre Nnn y tipo S2 la clase C1 asociada al esquema S1 debe tener dos métodos de acceso de la siguiente forma:

  public void setNnn(C2 c);
  public C2 getNnn();
  donde C2 es la clase asociada al esquema S2. Si S2 es un esquema definido en BasicOntology, entonces:

- Si S2 es el esquema de STRING, entonces C2 es java.lang.String.
- Si S2 es el esquema de INTEGER, entonces C2 es int, long, java.lang.Integer o java.lang.Long .
- Si S2 es el esquema de BOOLEAN, entonces C2 es boolean o java.lang.Boolean.
- Si S2 es el esquema de FLOAT, entonces C2 es float, double, java.lang.Float o java.lang.Double.
- Si S2 es el esquema de DATE, entonces C2 es java.util.Date.
- Si S2 es el esquema de BYTE_SEQUENCE, entonces C2 es byte[].
- Si S2 es el equema de AID (identificador de agente), entonces C2 es jade.core.AID
- Por cada campo en el esquema S1 con nombre nnn, tipo S2 y cardinalidad mayor que 1 la clase C1 asociada al esquema S1 debe tener dos métodos de acceso de la siguiente forma:
  public void setNnn(jade.util.leap.List l);
  public jade.util.leap.List getNnn();

Por tanto, en el ejemplo, las clases asociadas a los esquemas definidos en la ontología quedarían de la siguiente forma:

Url.java
ConexionA.java
Conecta.java

Para una información más detallada se puede consultar el documento *Jade Support to application-defined ontologies and content languages*, incluido en la documentación de JADE.

Mensajes y ontología

A la hora de que se comunican los agentes, es útil que estos empleen un lenguaje común referido a la realidad que los rodea. En estos casos, es cuando se utiliza el concepto de ontología, explicado anteriormente.
Jade permite el intercambio de mensajes utilizando información referida a una cierta ontología. Por ejemplo, si se hiciera uso de la conexionOntology, definida anteriormente, se enviaría un mensaje cuyo contenido fuera:

(Url ip:192.168.4.201 puerto:5004)

Esto es transformado por Jade en una cadena que se incluye en el contenido del mensaje por transmitir. Esta conversión y operaciones de comprobación son llevadas a cabo por un objeto content manager, que se encuentra en

el interior de cada agente y se accede a él mediante el método getContentManager() de la clase Agent.

El content manager provee de métodos para acceder a la funcionalidad de conversión, pero esta funcionalidad suele ser delegada en una ontología –una instancia de la clase Ontology incluida en jade.content.onto– y en un content language codec –una instancia del interfaz Codec contenido en jade.content.lang–. Más específicamente, la ontología valida la información que va a ser convertida desde un punto de vista semántico, mientras que el codec realiza la traducción a cadenas siguiendo las reglas sintácticas del content language.

Cuando se crea un agente que haga uso de una determinada ontología se hace uso de los métodos registerLanguage() y registerOntology del content manager del agente. Por ejemplo, en el caso de la conexionOntology, si se utiliza el codec SL, primero se debe crear las variables adecuadas dentro de la clase del agente:

```
private Codec codec = new SLCodec();
private Ontology ontologia =
PlataformaOntologia.getInstance();
```

(después de haber importado los paquetes correspondientes). Lo siguiente sería introducir el siguiente código dentro del método setup del agente:

```
getContentManager().
registerLanguage(codec);

getContentManager().
registerOntology(ontologia);
```

Esto se debe hacer tanto con los agentes que vayan a enviar tanto como con los que vayan a recibir mensajes que hagan uso de dicha ontología. Para enviar un mensaje se tiene que utilizar un código similar al siguiente dentro del comportamiento correspondiente:

```
try {
ACLMessage msg=newACLMessage(ACLMessage.
INFORM);
 msg.addReceiver(new AID((String)"nombre",false));
 msg.setLanguage(codec.getName());
 msg.setOntology(ontologia.getName());
 Url url = new Url();
 url.setIp("192.168.4.201");
 url.setPuerto(5004);
 ConexionA con = new ConexionA(); con.setUrl(url);
 getContentManager().fillContent(msg, con);
 send(msg);
}
catch (jade.content.lang.Codec.CodecException ce)
{
 System.out.println(ce);
}
catch (jade.content.onto.OntologyException oe) {
 System.out.println(oe);
}
```

Lo primero que se hace en este ejemplo es crear el mensaje ACL e indicar el nombre de su agente destinatario. Después indicamos la ontología y codec por utilizar, que se corresponderán con las instancias declaradas anteriormente.

En este ejemplo, se desea transmitir una IP y un puerto, comprendidos dentro de un hecho URL. Cada hecho se corresponde con una clase. Por tanto, se crea una instancia de dicha clase y se rellenan los campos correspondientes con los valores que se quieran enviar. Sin embargo, no se puede enviar un hecho directamente, pues un hecho no implementa la interfaz contentElement. Se debe enviar esa URL en el interior de un predicado ConexionA que le indique al agente de destino a la IP y al puerto a los que se debe conectar. Ese predicado se corresponde con otra clase definida con la ontología. Por último, se uti-

liza el content manager del agente para traducir ese predicado a una cadena, rellenando el contenido del mensaje con el método fillContent, y se envía el mensaje.
En el caso del agente receptor se puede utilizar un código similar al siguiente:

```
MessageTemplate mt = MessageTemplate.and(
MessageTemplate.MatchLanguage(codec.
getName()),
MessageTemplate.MatchOntology(ontologia.
getName()));
ACLMessage msg = receive(mt);
ACLMessage reply = msg.createReply();
if(msg.getPerformative()== ACLMessage.INFO){
ContentElement ce = getContentManager().
extractContent(msg);
 if (ce instanceof ConexionA) {
 ConexionA con = (ConexionA) ce;
 Url url = con.getUrl();
 System.out.println(«Conexion recibida: «);
 System.out.println(«URL: « + url.getIp());
 System.out.println(«PUERTO: « + url.getPuerto());
 }
 else {
reply.setPerformative(ACLMessage.
NOT_UNDERSTOOD);
 reply.setContent("(esperaba Movimiento)");
 send(reply);
 }
}
else {
 // Recibida una performativa no manejada por el
agente
reply.setPerformative(ACLMessage.
NOT_UNDERSTOOD);
  reply.setContent("( (Unexpected-act "+ACLMes-
sage.getPerformative(msg.getPerformative())+") (
expected (request :content ping)))");
 send(reply);
}
```

Al contrario de ejemplos anteriores, en este caso, como prerrequisito para la recepción de un mensaje este se debe adecuar a la ontología y al codec empleados. Para ello, se utiliza una instancia de la clase MessageTemplate que se le puede pasar como parámetro al método receive(). Una vez que se comprueba que la performativa del mensaje es la adecuada se extrae el ContentElement pasado como contenido del mensaje, utilizando el content manager. Se comprueba que dicho ContentElement es una instancia de la clase ConexionA, que es el predicado que está esperando el agente en este ejemplo. Por último, se extraen los datos del objeto URL que se obtienen del objeto ConexionA y en este caso, se muestran por la salida estándar.

Conclusiones

Desde que surgieron los trabajos distribuidos, un principal problema ha sido la administración de sus transacciones, toda vez que los lenguajes de programación como C, Java, etc. han resultado difíciles programar, principalmente, porque no ha habido una estandarización en esta problemática.
El nuevo paradigma de programación iniciado con Jade, el cual, ha sido un proyecto conjunto de varios grupos de personas, desde su estandarización hasta las herramientas para su implementación, ha dado soluciones inimaginables en el mundo de la programación distribuida, ya que primeramente Jade se encarga de toda la parte de comunicación, tanto de los protocolos como de los agentes, dejando a los programadores exclusivamente la parte sustancial del sistema, evitando la parte problemática en los lenguajes anteriormente descritos.

Jade ya incluye el administrador de transacciones desde su entorno de plataforma, lo que le facilita a los programadores evitar los problemas característicos en los procesos distribuidos, como es la colisión, redundancia, etc. Es importante aclarar que, para iniciar con Jade, es conveniente saber las espe-

cificaciones de la FIPA, toda vez que ahí se describen todas las actividades cotidianas en las transacciones distribuidas y que, por ser comunes, se han podido programar en Jade, además de las funciones específicas que tienen los agentes.

Las transacciones *federadas* que son transacciones realizadas desde diferentes ambientes operativos, diferentes equipos y diferentes bases de datos, son soportadas por Jade, ya que funciona como un *middleware*, tomando como base el Lenguaje Java para su operación. Esto quiere decir, que para que se ejecute Jade, primeramente, debe estar instalado Java en la computadora, y si se parte del hecho de que Java es multiplataforma, en consecuencia, Jade también lo es.

Trabajos futuros

Este trabajo, solamente abre la puerta de una inmensidad de aplicaciones que se pueden desarrollar; hasta donde se tenga conocimiento, los trabajos con Jade en México son insipientes y específicamente en Guerrero, no hay conocimiento que ya se esté trabajando con ello. Por tal razón, se pretende realizar una aplicación multiplataforma.

En un trabajo más específico, se puede implementar una aplicación de base de datos distribuida. Otro tema de interés está en hacer aplicaciones basadas en tecnología móvil, toda vez que los agentes tienen una característica muy especial para estos trabajos.

Referencias

Chavez (1996). An agent marketplace for buying and selling goods. Proceedings of first International Conference on the Practical Application of Intelligent Agents and Multi-Agent Systems, London, UK.

FIPA (2002). Recuperado de: http://www.fipa.org.

Gómez y Zea (2011). Incorporación de agentes inteligentes en ambientes de Aprendizaje.

Hewitt, C. (1977). Actor Model.

Huhns and Singh (eds.) (1997). *Agents and Multiagent Systems: Themes, Approaches, and Challenges. Reading in Agents...* Ciudad: Morgan Kaufmann Publishers.

JADE (2004). Recuperado de: http://www.tilab.it.jade.com

Jennings (2000). *Agent Oriented Software Engineering. Handbook of Agent Technology* (ed. J.Bradshaw) AAAI/MIT Press (en prensa).

Krulwich (1996). *The BargainFinder agent: Comparison price shopping on the internet*. Ciudad: J. Williams, editor, Bots, and other Internet Beasts. Macmillan Computer Publishing: Indianapolis.

Lashkari (1994). Collaborative interface agents. Proceedings of the National Conference on Artificial Intelligence. MIT Press, Cambridge, Mass.

Mitchell (1994). Experience with a Learning Personal Assistant. *Communications of the ACM*, *37*.

Pelagatti, Politecnico di Milano y Ceri, S. (1985). *Distributed Databases. Principles and Systems*. Ciudad: International Student Edition. McGaw-Hill Book Company,

Riecken (1994). Architecture of integrated agents. *Communications of the ACM, 37.*

Selker (1994). A teaching agent that learns. *Communications of the ACM, 37.*