MANUAL DE JADE PARA PRINCIPIANTES

ALEJANDRO FRANCO BARRIOS

UNIVERSIDAD TECNOLOGICA DE BOLIVAR

FACULTAD DE INGENIERIA

DIRECCION DE PROGRAMA DE INGENIERIA DE SISTEMAS

CARTAGENA DE INDIAS D.T Y C.

2009

MANUAL DE JADE PARA PRINCIPIANTES

ALEJANDRO FRANCO BARRIOS

MONOGRAFIA PRESENTADA COMO REQUISITO FINAL PARA APROBAR EL MINOR DE DESARROLLO DE APLICACIONES DISTRIBUIDAS

DIRECTOR

Msc. MOISES QUINTANA ALVAREZ

UNIVERSIDAD TECNOLOGICA DE BOLIVAR

FACULTAD DE INGENIERIA

DIRECCION DE PROGRAMA DE INGENIERIA DE SISTEMAS

CARTAGENA DE INDIAS D.T Y C.

2009

Cartagena de Indias D.T y C. Mayo 18 de 2009

Señores
UNIVERSIDAD TECNOLOGICA DE BOLIVAR
COMITÉ DE EVALUACION DE PROYECTOS
FACULTAD DE INGENIERIA DE SISTEMAS
CIUDAD

Estimados Señores,

De la manera más cordial me dirijo a ustedes, con el fin de presentarles a su estudio, consideración y aprobación la monografía que lleva por título **"MANUAL DE JADE PARA PRINCIPIANTES"**, como requisito final para aprobar el Minor de Desarrollo de Aplicaciones Distribuidas.

Espero que este proyecto sea de su total agrado.

Cordialmente,

_____

Alejandro Franco Barrios
C.C 1.047.373.924 de Cartagena

Cartagena de Indias D.T y C. Mayo 18 de 2009

Señores
UNIVERSIDAD TECNOLOGICA DE BOLIVAR
COMITÉ DE EVALUACION DE PROYECTOS
FACULTAD DE INGENIERIA DE SISTEMAS
CIUDAD

Estimados Señores,

Tengo el agrado de presentar la monografía, en la cual me desempeño como director, titulada **"MANUAL DE JADE PARA PRINCIPIANTES"**, desarrollada por ALEJANDRO FRANCO BARRIOS, como requisito final para aprobar el Minor de Desarrollo de Aplicaciones Distribuidas.

Considero que el trabajo es satisfactorio y amerita ser presentado para su evaluación.

Cordialmente,

_____

Moisés Quintana Álvarez

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

_____

FIRMA DEL PRESIDENTE DEL JURADO

_____

FIRMA DEL JURADO

_____

FIRMA DEL JURADO

Cartagena de Indias D.T y C. Mayo 18 de 2009

AUTORIZACION

Yo ALEJANDRO FRANCO BARRIOS, identificado con la cedula de ciudadanía numero 1.047.373.924 de Cartagena, autorizo a la Universidad Tecnológica de Bolívar, para hacer uso de mi monografía y publicarla en el catalogo online de la biblioteca.

_____

Alejandro Franco Barrios
C.C 1.047.373.924 de Cartagena

A Dios por darme la fuerza y el valor para culminar con mi carrera; a mi familia por apoyarme incondicionalmente y soportarme en mi camino de crecimiento profesional y como persona, y a todos aquellos que siempre estuvieron ahí colaborándome de una forma u otra.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE ILUSTRACIONES

INTRODUCCION

Hoy en día vivimos en una gran revolución tecnológica, tanto en el mundo de las comunicaciones como en el de los sistemas de información. Esta revolución es constituida básicamente, por el crecimiento en la utilización del internet, los sistemas de información, juegos, animaciones, comercio electrónico, plataformas de educación virtual, los avances alcanzados hace que cada vez cobre más fuerza la idea de que nos encontramos inmersos en una sociedad de la información. En consecuencia, para el manejo de toda esta cantidad de datos y con el objeto de facilitar al usuario final el uso de las herramientas informáticas distribuidas, han surgido los comúnmente denominados ***Agentes***, también conocidos como *Agentes inteligentes, de software, Autónomos o Softbots*.

Los agentes inteligentes se están utilizando en una gran variedad de aplicaciones, debido a las ventajas que ofrecen, entre las que destaca las consultas, el filtrado, la recuperación, organización y mantenimiento de la información relevante para el usuario final. Varios agentes pueden conformar un Sistema Multi-agente, capaces de interactuar en un entorno común; de esta forma poseen capacidades como la comunicación, negociación, y coordinación entre los agentes que lo conforman. También se pueden considerar características opcionales como la movilidad, la necesidad de interacción con usuarios y el consiguiente aprendizaje de su comportamiento.

JADE (Java Agent Development Framework), es una plataforma desarrollada íntegramente en Java por TILAB, que proporciona tanto un entorno de desarrollo como un entorno de ejecución para la creación de Sistemas Multi-agente.

El entorno de desarrollo de Jade está formado por una serie de librerías en Java que permiten la implementación de agentes de manera limpia e independiente de la plataforma sobre la que se va a ejecutar.

El entorno de ejecución permite a los agentes existir y comunicarse entre ellos, proporciona una serie de herramientas que permiten al desarrollador controlar y depurar a los agentes en tiempo real.

Jade responde a los estándares FIPA (Foundation for Intelligent Physical Agents) que es una organización que se encarga de desarrollar especificaciones estándar para los sistemas basados en agentes. Esto es para facilitar la interconexión entre plataformas de diferentes empresas y organizaciones. El objetivo FIPA es definir la interoperabilidad entre los sistemas basados en agentes, dejando fuera la implementación interna.

FIPA define el modelo de una plataforma basada en agentes, el conjunto de servicios que debe proveer y la interface entre los servicios.

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).

RESUMEN

http://jade.tilab.com/

Jade, significa Java Agent Development Framework y es la implementación oficial del estándar FIPA (Foundation for Intelligent Physical Agents), soporta todos los servicios básicos de infraestructura especificados en FIPA, a los que añade algunas utilidades gráficas para facilitar la administración de las plataformas y la depuración de los mensajes intercambiados por agentes en tiempo de ejecución.

Esta es una herramienta para el desarrollo de sistemas multi-agente, y está totalmente implementada en Java. Esto facilita la portabilidad y la movilidad de los agentes entre distintas plataformas, incluso si no están desarrolladas en Jade, solo es necesario que estas plataformas estén regidas por las especificaciones de FIPA.

Teniendo en cuenta la tendencia actual de los sistemas de software a ser concurrentes y distribuidos, además de tener la capacidad de encontrar dinámicamente los servicios de red y permanecer siempre en marcha, o en ejecución, no se pueden utilizar técnicas tradicionales de programación, como puede notarse hoy día en lo que se denomina la red semántica. Es por esto que Jade utiliza inteligencia artificial, más específicamente los agentes inteligentes, que poseen ciertos comportamientos que funcionan en un entorno o ambiente, reaccionan frente a cambios que se puedan presentar y toman sus propias decisiones con base en lo percibido y a su estado interno.

La comunicación entre agentes se lleva a cabo a través de mensajes asíncronos, es decir, el agente que envía el mensaje y el destinatario del mensaje no tienen porque estar disponibles al mismo tiempo, ni siquiera el destinatario tiene que existir en ese

instante. La estructura de los mensajes se basa en el lenguaje ACL (Agent Communication Language) que ha sido definido por FIPA.

Jade posee una cantidad de comandos que se usan para su administración, estos comandos se ejecutan desde una consola, o desde una interfaz gráfica que es llamada RMA. Para empezar a programar agentes se debe tener en cuenta que hay que definirles unos comportamientos y unas ontologías dependiendo del ambiente en que estén inmersos.

OBJETIVOS

General

Proporcionar la explicación básica e ilustrar el proceso de iniciación de quienes deseen conocer la herramienta JADE para la manipulación y el control de agentes inteligentes.

Específicos

- Enseñar los conceptos básicos acerca de agentes y sistemas multi-agente.
- Dar las pautas para empezar la programación en Jade.
- Enseñar la correcta instalación de Jade a través de la línea de comandos.
- Dar a conocer ciertos comandos para facilitar el uso y manejo de los agentes.
- Mostrar la forma de ejecutar los agentes y las herramientas gráficas que ofrece Jade.

CAPITULO I. CONCEPTOS BASICOS

1.1 INTRODUCCION.

Este capítulo presenta los conceptos básicos acerca de Jade, los agentes, los tipos existentes, sus comportamientos, su interacción con el ambiente en que operan, su manera de ser sociables y todo lo necesario para su comprensión.

No existe una definición universalmente aceptada de lo que es un agente, pero existe un consenso general en la comunidad informática sobre su característica principal que es la de poseer autonomía, del resto de atributos hay algunas contrariedades.

Un agente es un sistema computacional capaz de actuar independientemente del usuario. Pero esta definición deja muchas dudas en el aire ya que es muy general.

1.2 JADE.

Es un middleware para el desarrollo de aplicaciones multi-agente distribuidas, basado en una arquitectura con comunicación peer-to-peer (P2P).

Además de proporcionar un API para la creación de agentes y elementos relacionados, pone a nuestra disposición una interfaz gráfica y una serie de herramientas para el control y la depuración de nuestro sistema durante el desarrollo del mismo. Además se 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.

Jade presenta las siguientes características:

- P2P: Arquitectura peer-to-peer, cada agente puede tomar la iniciativa en una comunicación o bien responder a peticiones que le hagan otros agentes.

- Interoperabilidad: Jade cumple con las especificaciones FIPA, por lo que los agentes desarrollados en Jade pueden interactuar con otros agentes que no

tienen porque estar desarrollados con Jade, aunque si deben seguir las especificaciones FIPA.

- Portabilidad: La API que proporciona Jade es independientemente de la red sobre la que va a operar, así como de la versión de Java utilizada, teniendo la misma API para J2EE, J2SE y J2ME[1].

- Intuitiva: Jade se ha desarrollado para ofrecer una API fácil de aprender y sencilla de manejar.

Los agentes Jade tienen nombres únicos y se permite a cada agente descubrir a otros agentes y comunicarse con ellos mediante comunicaciones punto a punto. Los agentes proporcionan servicios, cada agente puede buscar a otros dependiendo de los servicios que proporcionen otros agentes. La comunicación entre agentes se lleva a cabo a través de mensajes asíncronos, es decir, el agente que envía el mensaje y el destinatario del mensaje no tienen porqué estar disponibles al mismo tiempo. Es más, el destinatario no tiene porqué existir en ese instante. Los mensajes se pueden enviar a un agente en concreto o se pueden enviar a agentes que se desconocen pero se sabe que poseen unas ciertas características. Jade proporciona mecanismos de seguridad, ya que habrá agentes a los que no se les esté permitido comunicarse con otros agentes, de manera que una aplicación puede verificar la identidad del receptor y del agente que envía el mensaje y no dejar realizar actuaciones no permitidas para un determinado agente. La estructura de los mensajes se basa en el lenguaje ACL (Agent Communication Language) que ha sido definido por la FIPA. Jade también permite a los agentes cambiar de Host (en J2SE). Un agente puede interrumpir en un momento dado su ejecución, migrar a otro host (sin necesidad de que el código esté previamente en el host destino) y continuar la ejecución en el mismo punto en el que la interrumpió. Esto

[1] J2EE: Java 2 Enterprise Edition, J2SE: Java 2 Standart Edition, J2ME: Java 2 Micro Edition.

permite un balanceo de carga ya que permite a los agentes migrar a hosts menos cargados. El entorno de ejecución proporciona un marco donde poder ejecutar los agentes y herramientas gráficas para su monitorización y depuración.

1.2.1 ARQUITECTURA.

El entorno de ejecución de Jade es donde los agentes pueden “vivir”. Una instancia de ese entorno se denomina *Contenedor*, y es posible albergar en él un número indeterminado de agentes. Al conjunto de contenedores se les denomina *plataforma,* esta proporciona una capa que oculta a los agentes el entorno donde se ha decidido ejecutar la aplicación, un servicio de páginas blancas para buscar otros agentes, un servicio de páginas amarillas para buscar servicios que otros agentes ofrecen, un modulo de gestión a través del cual se accede a estas facilidades y un sistema de envió/recepción de mensajes entre agentes.

Existe un tipo de contenedor especial denominado *principal*. Debe existir uno y solo uno de estos contenedores por cada plataforma FIPA de agentes y el resto de contenedores de la plataforma una vez ejecutados deben suscribirse al principal, por lo que el responsable de ejecutarlos también es responsable de indicar en dónde se localiza el contenedor principal. La principal diferencia del contenedor principal respecto al resto de contenedores, es que alberga agentes especiales:

- **AMS (Agent Management System):** Este agente proporciona el servicio de nombres asegurando que cada agente en la plataforma disponga de un nombre único. También representa la autoridad, es posible crear y matar agentes en contenedores remotos por medio del agente AMS, de igual manera se encarga de hacer llegar a su destino los mensajes generados por los agentes ubicados en la misma plataforma, a través de un sistema de envió/recepción de mensajes.
- **DF (Directory Facilitator):** Proporciona el servicio de *Páginas amarillas,* por lo tanto un agente puede encontrar otros agentes que provean los servicios necesarios para lograr sus objetivos.

- **ACC (Agent Communication Channel)**: software que controla el intercambio de mensajes.

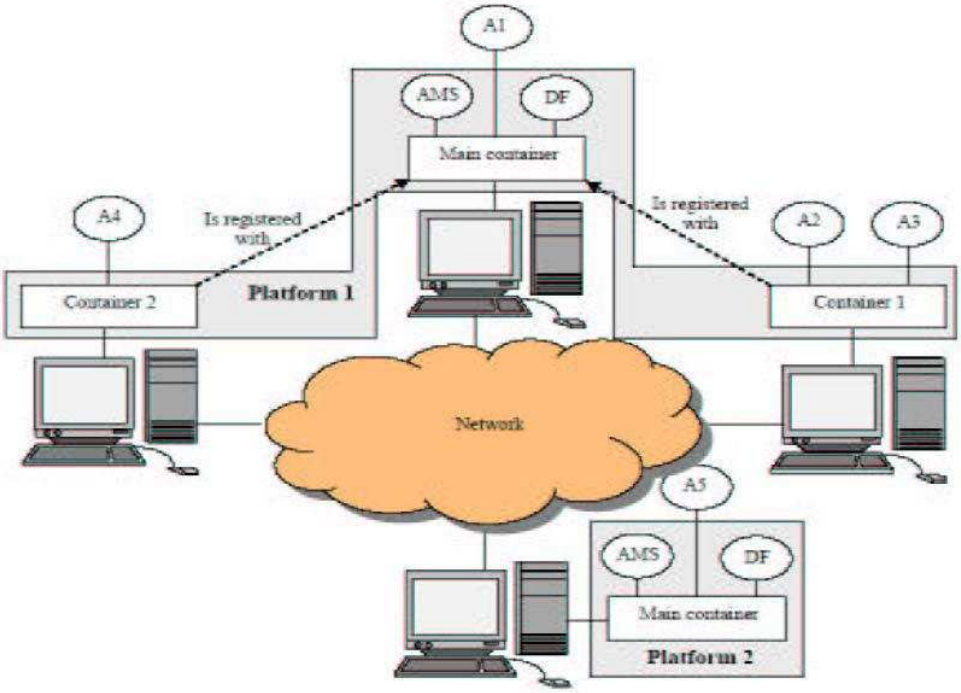


Ilustración 1 ARQUITECTURA DE JADE

En la ilustración 1, se puede ver que en la plataforma número 1 existen tres contenedores. En el principal están los elementos imprescindibles el AMS y el DF. También existe el agente A1. Dentro de esta misma plataforma hay otros dos contenedores normales, el de la izquierda albergando al agente A4, y el de la derecha a los agentes A2 y A3. La segunda plataforma está únicamente formada por un contenedor principal, y alberga también al AMS y al DF, junto con al agente A5. Todos los agentes de la ilustración 1 pueden comunicarse, siempre y cuando sepan el nombre del agente con quien quieren hacerlo.

1.2.2 LA COMUNICACIÓN EN JADE.

Maneja el lenguaje de comunicación de agentes ACL, con un mecanismo de paso de mensajes asíncrono, cada agente tiene un buzón en el cual se van almacenando los mensajes enviados por otros agentes, es una comunicación a temporal (cola de mensajes), donde hay interlocutores heterogéneos y una traducción a diversos estándares como XML[2], tiene una biblioteca de *protocolos de interacción*[3], que cuenta con un esqueleto de modelos típicos de interacción, donde se puede dar una negociación, votación y delegación de tareas.

1.2.3 BONDADES DE JADE.

- Administración del ciclo de vida de los agentes.
  - Dinámicamente conocer los agentes activos.
  - Lanzar, monitorizar, suspender y matar agentes.
  - Movilidad de agentes.
- Provee agentes para la administración de la plataforma.
  - Interfaz gráfica de usuario (GUI).
  - Monitorear las comunicaciones.
- Servicio de páginas amarillas
  - Registro y búsqueda de servicios.
- Los agentes son organizados en contenedores y plataformas.
  - Los contenedores alojan agentes.
  - Las plataformas tienen contenedores.
  - Cada plataforma tiene un contenedor principal.
- Servicio de nombres.
  - Nombres únicos.
- Seguridad.

[2] XML: Extensible Markup Language.
[3] Protocolos de interaccion: ver la tabla 2.

- Autenticación de usuarios y agentes.
- Asignación de derechos.

1.3 AGENTES.

Existen muchas definiciones acerca de lo que es un agente, pero una de gran aceptación es la que está dada por Pattie Maes, investigadora del MIT[4]: "un agente es un sistema computacional que vive en un entorno complejo y dinámico. El agente puede sentir ese entorno y actuar en consecuencia, y tiene un conjunto de objetivos o motivaciones que intenta conseguir a través de dichas acciones".

Como he mencionado anteriormente, la definición de agente no es una tarea fácil, por el contrario, ha sido un aspecto que ha suscitado un amplio debate en la comunidad científica. Un agente es un sistema informático, situado en algún entorno, dentro del cual es capaz de realizar acciones de forma autónoma y flexible para así cumplir sus objetivos. Un agente recibe entradas sensibles de su entorno y a la vez ejecuta acciones que pueden cambiar este entorno [Russell 1995]. Es ampliamente aceptada la caracterización de un agente [Woldridge 1995] como aquel sistema informático que satisface las siguientes propiedades (las cuatro primeras se consideran básicas y el resto opcionales):

- ***Autonomía*:** tiene la capacidad de actuar sin intervención humana directa o de otros agentes.
- ***Sociabilidad*:** capacidad de interactuar con otros agentes, utilizando como medio, algún lenguaje de comunicación entre agentes.
- ***Reactividad*:** un agente está inmerso en un determinado entorno o ambiente del que percibe estímulos y ante los que debe reaccionar en un tiempo preestablecido.

[4] MIT: El Instituto Tecnológico de Massachusetts, una de las principales instituciones dedicadas a la investigación en Estados Unidos.

- ***Iniciativa*:** un agente no sólo debe reaccionar a los cambios que se produzcan en su entorno, sino que tiene que tener un carácter emprendedor y tomar la iniciativa para actuar guiado por los objetivos que debe satisfacer.
- ***Movilidad*:** habilidad de un agente de trasladarse en una red de comunicación informática.
- ***Veracidad*:** propiedad por la que un agente no comunica información falsa intencionadamente.
- ***Benevolencia*:** un agente no tiene objetivos contradictorios y siempre intenta realizar la tarea que se le solicita.
- ***Racionalidad*:** un agente tiene unos objetivos específicos y siempre intenta llevarlos a cabo.

Un agente no tiene un control total sino parcial del ambiente en el que se desempeña, razón por la cual el resultado de una acción ejecutada no siempre es el esperado. Generalmente por esta razón se denomina a los ambientes con mediana complejidad como no determinísticos.

Para modificar un ambiente, un agente posee un repertorio de acciones, pero cada una de ellas es aplicable solo en una situación definida, es decir, para su ejecución cada acción depende de unas precondiciones asociadas a ella. Un ejemplo muy utilizado para describir el comportamiento de un agente es un termostato.

El termostato tiene un sensor para detectar la temperatura de una habitación, de manera que es capaz de producir una señal de salida cuando la temperatura es baja o está dentro del límite aceptado. El termostato tiene un conjunto de acciones para ejecutar de acuerdo al estado de la señal que el sensor esté produciendo, así, tenemos un conjunto de precondiciones y uno de acciones a ejecutar en cada caso de la siguiente manera:

PRECONDICIÓN	ACCIÓN
Temperatura baja	Subir la calefacción
Temperatura alta	Apagar la calefacción

Tabla 1 PRECONDICIONES Y ACCIONES DE UN AGENTE (TERMOSTATO).

1.3.1 AMBIENTES.

El comportamiento de un agente es afectado fuertemente por el ambiente en el cual esta embebido, de las siguientes maneras:

- El agente percibe directamente el ambiente mediante sus sensores.
- El agente actúa mediante sus efectores modificando el ambiente y sus percepciones futuras.
- Corporalidad: caso especial en el cual el agente tiene un cuerpo físico que restringe su interacción con el ambiente.

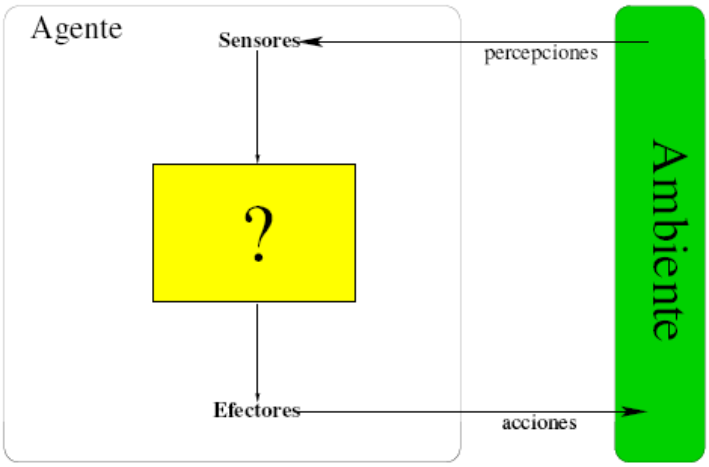


Ilustración 2 EL AGENTE ES AFECTADO POR EL AMBIENTE.

1.3.1.1 ACCESIBLE.

Un ambiente accesible es aquel en el cual un agente puede obtener información completa, exacta y actualizada del estado del ambiente. Muchos ambientes del mundo real no son accesibles, tal es el caso de Internet.

1.3.1.2 DETERMINISTICO

Un ambiente determinístico es aquel en el cual la ejecución de una acción tiene un efecto garantizado, sin incertidumbre sobre el estado que tendrá resultado debido a la acción.

1.3.1.3 ESTATICO

Un ambiente estático es aquel con la propiedad de permanecer estable hasta que un agente ejecuta una acción.

En contraste, en un ambiente dinámico existen otros procesos operando, por lo tanto el agente no tiene control sobre él.

1.3.1.4 DISCRETO

Un ambiente es discreto si hay un número fijo y finito de acciones y percepciones en él.

También existen sus opuesto, o sea inaccesibles, no determinísticos, dinámicos y continuos. Un buen agente es aquel que es capaz de tomar decisiones. La calidad de estas decisiones depende de que tan bien informado esté sobre su ambiente. Si el nivel de información que posee a la hora de tomar una decisión es precario, entonces el resultado muy probablemente no será el esperado. Cuando hablamos de ambientes determinísticos decimos que el resultado de cualquier acción ejecutada es único. Esta propiedad es preferible desde el punto de vista de un diseñador de agentes ya que no existe ningún nivel de incertidumbre sobre el resultado de una acción. Desafortunadamente si un ambiente es lo suficientemente complejo, el hecho de que

sea determinístico no es de mucha ayuda. En la práctica, los ambientes deben ser asumidos como no determinísticos, ya que el efecto de una acción sobre el medio no puede ser cuantificada, este concepto se relaciona con el de dinamismo.

En los inicios de la investigación de Inteligencia Artificial se crearon algoritmos de planeación que dan una descripción del estado actual de un ambiente, las acciones disponibles dentro de este, los efectos que estas tienen y el estado esperado. En este tipo de algoritmos se genera un plan (una secuencia de acciones) de manera que cuando se ejecutan a partir del estado inicial, se alcanzará el estado objetivo. Sin embargo, este tipo de algoritmos fueron diseñados para funcionar únicamente en ambientes estáticos, es decir, aquellos que no cambian excepto cuando un agente actúa sobre ellos. Es claro que muchos ambientes del mundo real no son estáticos, tal es el caso de los sistemas operativos, en los cuales muchos procesos se están ejecutando concurrentemente en formas en las que un agente no tiene capacidad de controlar.

Desde el punto de vista de un agente, los ambientes dinámicos tienen dos importantes propiedades. La primera está relacionada con el tiempo de ejecución de una acción y el estado inicial de un ambiente, en otras palabras, un agente ejecuta una acción en un intervalo de tiempo determinado t1 – t2, en el transcurso del cual el estado inicial del ambiente puede cambiar. Esto significa que mientras un agente ejecute una acción debe recolectar información para determinar el estado actual del ambiente ya que sus condiciones iníciales pueden cambiar en el intervalo t1 – t2 y la meta lograda puede ser diferente a la esperada [Moore 1990].

La segunda propiedad está relacionada con la interferencia que otros procesos pueden producir sobre la acción que se pretende ejecutar. De esta manera un agente no puede estar seguro de que la acción que ejecutará tendrá el resultado que espera. Las propiedades descritas para ambientes dinámicos y estáticos sugieren que a nivel de diseño son más manejables los segundos que los primeros, ya que en estos el agente

necesita recolectar información sobre su ambiente una sola vez y las acciones que ejecute tendrán los efectos deseados. Además, un agente que opere en un ambiente estático no debe preocuparse por asuntos como la coordinación y sincronización de sus acciones con otros procesos ejecutándose en el mismo ambiente [Bond 1988].

Para finalizar el análisis de los ambientes, tenemos una última relación entre los discretos y los continuos. Un ambiente discreto es aquel que posee un número finito de estados; en contraste en un ambiente continuo existen infinitos estados posibles. Según lo anterior, podemos concluir que el diseño de agentes en un estado discreto es más simple que en uno continuo por varias razones, una de ellas es que los computadores son un ambiente de este tipo, aunque muchas veces pueden simular ambientes de tipo continuo realizando procesos de muestreo para transformar este tipo de ambientes a discretos, obteniendo como resultado una aproximación de ellos; de esta manera la información que un agente recolecte de un ambiente discreto producido a partir de un ambiente continuo no será completamente exacta.

En general un ambiente discreto brinda la posibilidad de enumerar todos los posibles estados y las acciones a ejecutar en cada uno de ellos, a diferencia de lo que ocurre con los ambientes continuos.

1.3.2 AGENTES EN TIEMPO REAL.

Un aspecto relacionado con la interacción de un agente con su ambiente es el concepto de tiempo real. Este concepto juega un papel importante cuando tenemos en cuenta el tiempo de ejecución de las acciones de un agente. Se pueden identificar diferentes tipos de interacciones de tiempo real:

- Las que definen que acciones deben ser ejecutadas en un tiempo determinado.
- Aquellas en las cuales el agente debe efectuar una acción tan rápidamente como pueda.

- En las que el agente debe repetir alguna tarea a una frecuencia determinada.

1.3.3 EL AGENTE RACIONAL.

En el diseño de agentes artificiales se busca que el agente tenga buen comportamiento.

El agente racional, intuitivamente, es aquel que siempre realiza la acción correcta.

Una acción correcta, es aquella que causa que el agente sea más exitoso.

Medida de desempeño (performance): criterio que establece cuan exitoso es el comportamiento de un agente.

La definición de un agente racional [Russell 1995] es:

“Para cada secuencia de percepciones posibles, un agente racional debería seleccionar aquella acción que se espera que maximice su medida de performance, dada la evidencia provista por la secuencia de percepciones y cualquier conocimiento previo que el agente tenga”.

1.3.4 AGENTES INTELIGENTES Y EVENTOS.

Cuando un evento ocurre, modifica el ambiente y esto es algo de lo cual el agente debe estar consciente. Los agentes pueden ejecutar acciones para hacer cosas por nosotros. Aunque, el hecho de que sistemas computacionales hagan cosas por nosotros no es nada nuevo, en los sistemas convencionales no tenemos un nivel de confianza que nos asegure que los comandos que ejecutamos produzcan los resultados que tenemos en mente. Cuando es un agente quien realiza alguna acción por nosotros, el nivel de confianza en que se ejecutará lo que pensamos aumenta debido a que en este caso el proceso tiene características de racionalidad que el agente le aporta y va de acuerdo con nuestros intereses. Como en todas las situaciones, cuando delegamos acciones a

terceros corremos riesgos pero también obtenemos beneficios. En este caso el riesgo es que el agente confunda sus metas y nos tome más trabajo reparar el daño causado, pero por otro lado se encuentra la ventaja de estar libres de preocupaciones sobre la ejecución de una actividad determinada [Bigus 2001].

1.3.5 CLASIFICACION DE LOS AGENTES.

Los agentes cuentan con diferentes formas de clasificación, precisamente porque es muy difícil para los investigadores ponerse de acuerdo en la definición, ahora en la clasificación ocurre lo mismo.

Se pueden clasificar según su delegación, inteligencia y movilidad. Otra forma está enfocada hacia la estrategia de procesamiento del agente. Y por último, según la acción que ejecuta.

1.3.5.1 DELEGACION, INTELIGENCIA Y MOVILIDAD.

Delegación tiene que ver con el grado de autonomía que el agente software tiene para representar al usuario ante otros agentes, aplicaciones, y sistemas computacionales, en otras palabras, el agente es un delegado del usuario. Un agente representa a un usuario, lo guía, lo ayuda y en algunos casos, toma decisiones unilaterales para beneficiarlo.

La Inteligencia se refiere a la habilidad del agente para capturar y aplicar un dominio específico de conocimiento para resolver problemas. De esta manera los agentes pueden ser relativamente simples cuando utilizan para este propósito lógica codificada, o pueden ser relativamente sofisticados si utilizan métodos basados en Inteligencia Artificial tales como Inferencia y Aprendizaje.

La movilidad se enfatiza en las habilidades sociales y la autonomía. Los agentes móviles son procesos de software que son capaces de transitar por una red, generalmente una WAN[5], interactuando con computadores alejados, reuniendo información para el usuario y volviendo a su origen cuando las tareas fijadas por el usuario se hayan completado. Las tareas que se pueden realizar son por ejemplo reservaciones de vuelos, manejo de una red de telecomunicaciones entre otras.

1.3.5.2 ESTRATEGIAS DE PROCESAMIENTO.

- Agentes reflejos (o reactivos) simples.
- Agentes reflejos basados en modelos.
- Agentes basados en objetivos (o deliberativos).
- Agentes basados en utilidades.
- Agentes colaborativos.

1.3.5.2.1 AGENTES REFLEJOS SIMPLES.

Usan una serie de condicionales para decidir la acción a tomar.

También llamados agentes reactivos puros.

Seleccionan una acción con base en la percepción actual, ignorando el resto de la historia perceptual (el pasado).

No existe internamente ninguna representación de estado.

La decisión sobre la acción a tomar se basa en un conjunto de reglas condición-acción (o situación-acción).

[5] WAN: Wide Area Network o red de Area Amplia.

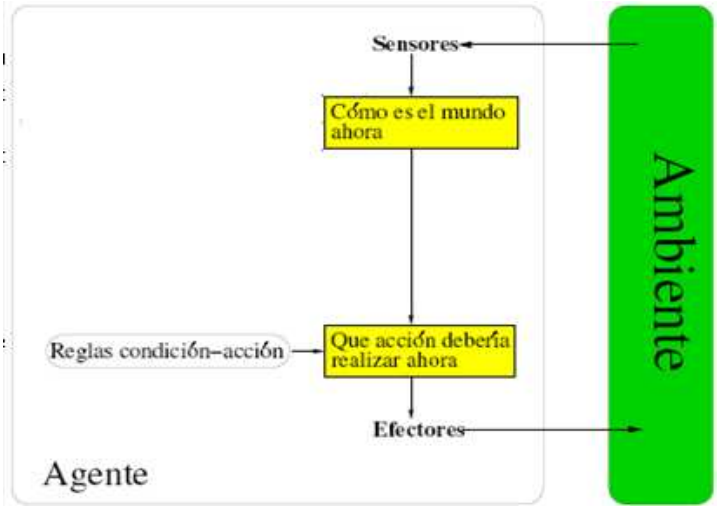


Ilustración 3 ESTRUCTURA DE UN AGENTE REFLEJO SIMPLE.

Ejemplo: si auto-adelante-esta-frenando entonces comenzar-a-frenar.

1.3.5.2.2 AGENTES REFLEJOS BASADOS EN MODELOS.

Es igual que el reflejo simple, pero conoce la repercusión de sus acciones.

- También llamados agentes reflejos con estado.
- Cuentan con alguna estructura de datos o estado interno que registra Información sobre el estado del ambiente y la historia perceptual.
- Actualizar la información de estado interno requiere un modelo del mundo:
  - Las acciones del agente modifican el mundo de alguna manera.
  - El mundo evoluciona independientemente del agente.
- Si bien toma en cuenta el pasado no considera el futuro (no planifica).

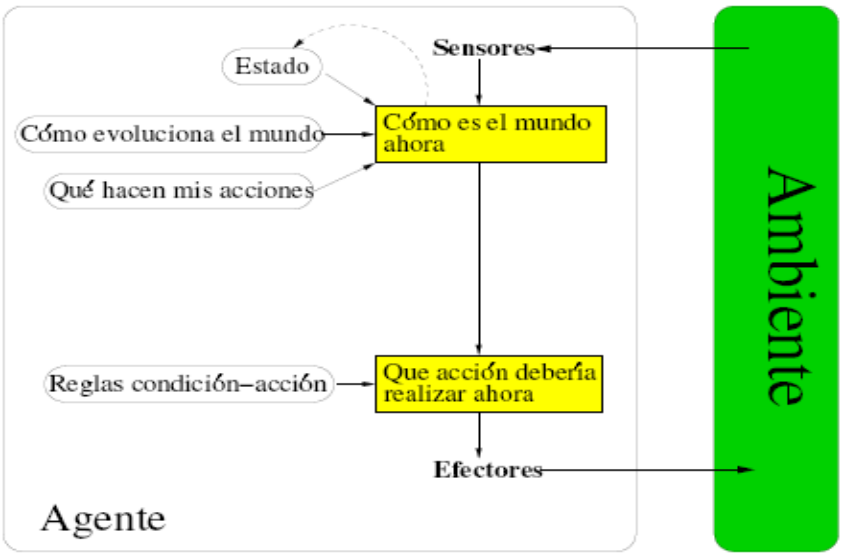


Ilustración 4 ESTRUCTURA DE UN AGENTE BASADO EN MODELOS.

1.3.5.2.3 AGENTES BASADOS EN OBJETIVOS O METAS.

- En la selección de acciones se toma en cuenta información sobre los objetivos (estados deseables) a alcanzar.
- El logro de un objetivo puede requerir analizar las consecuencias futuras de secuencias completas de acciones (planes).

Comparación respecto a los agentes reflejos:

- Son menos eficientes.
- Son más flexibles (cambios de objetivo y condiciones cambiantes).

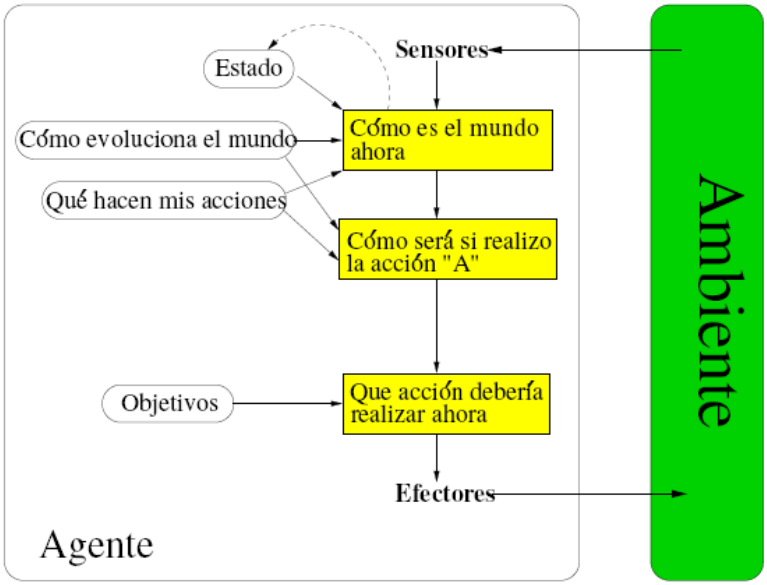


Ilustración 5 ESTRUCTURA DE UN AGENTE BASADO EN METAS

1.3.5.2.4 AGENTES BASADOS EN UTILIDADES.

- Permiten realizar una distinción más fina entre los objetivos o estados a alcanzar que los Basados en Objetivos.
- Si las metas (objetivos) se encuentran en conflicto, o se puede alcanzar más de una meta al mismo tiempo, se requiere conocer un valor que defina que meta es prioritaria, ese valor es la utilidad.
- Un Agente Basado en Utilidad es más adecuado que uno basado en objetivos cuando:
  - Es necesario balancear objetivos conflictivos.
  - Es necesario ponderar la importancia de varios objetivos para los que no existe certeza de ser alcanzados.

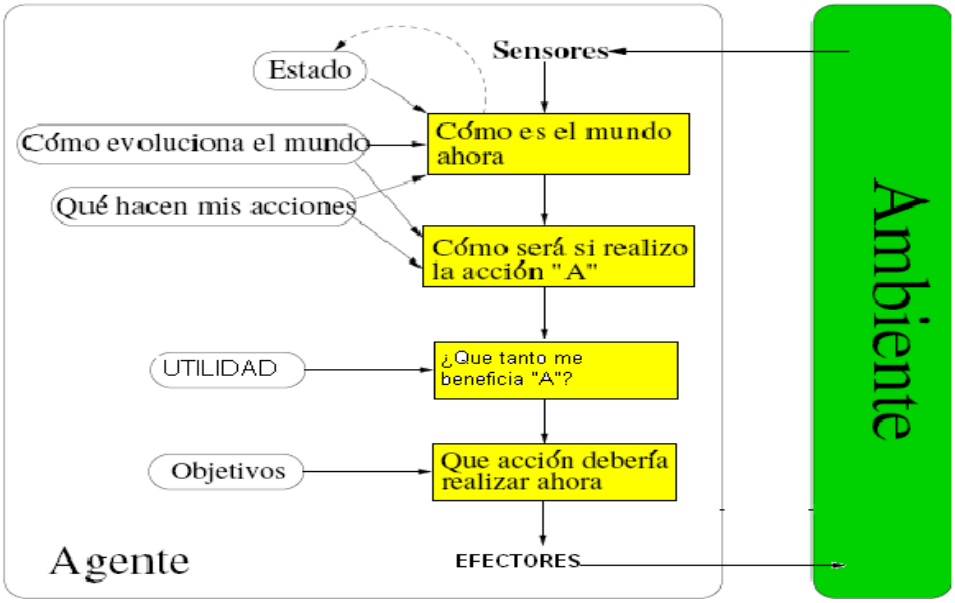


Ilustración 6 ESTRUCTURA DE UN AGENTE BASADO EN UTILIDADES.

1.3.5.2.5 AGENTES COLABORATIVOS.

Este tipo de agentes se enfatiza en la autonomía y las habilidades sociales con otros agentes para ejecutar las tareas de sus usuarios. La coordinación de los agentes se logra mediante la negociación para alcanzar acuerdos que sean aceptables para los agentes negociantes. Los agentes colaborativos son capaces de actuar racionalmente y autónomamente en ambientes multi-agente y con restricciones de recursos. Otras características de estos agentes es que poseen habilidades sociales, son proactivos, benévolos, estáticos y veraces.

Los agentes colaborativos se pueden utilizar en la solución de algunos de los siguientes problemas:

- Para resolver problemas que son muy grandes para un agente centralizado.

- Para permitir la interconexión e interoperabilidad de sistemas de IA existentes como sistemas expertos, sistemas de soporte de decisión etc.
- Solucionar problemas que son inherentemente distribuidos.
- Proporcionar soluciones que simulen recursos de información distribuidos.
- Incrementar la modularidad, velocidad, confiabilidad, flexibilidad y reutilización en sistemas de información.

1.3.5.3 FUNCIONES DE PROCESAMIENTO.

Los agentes se pueden clasificar también de acuerdo a la función que desempeñan, de la siguiente manera:

- Agentes de información.
- Agentes de interfaz.

1.3.5.3.1 AGENTES DE INFORMACION.

Los agentes de información nacieron debido a la gran cantidad de herramientas que surgieron para el manejo y recuperación de información. Los agentes de información tienen los roles de manejar, manipular, e integrar información de muchas fuentes de datos distribuidas.

La hipótesis fundamental de los agentes de información es que puedan mejorar de algún modo, pero no completamente el problema de la sobrecarga de información y en general el manejo de esta.

1.3.5.3.2 AGENTES DE INTERFAZ.

Trabajan como asistentes personales para ayudar a un usuario a ejecutar tareas. Este tipo de agentes usualmente emplean aprendizaje para adaptarse a los hábitos de trabajo y preferencias del usuario.

Se han identificado cuatro formas de aprendizaje utilizadas por este tipo de agentes:

- El agente observa el comportamiento del usuario y lo imita.
- El agente puede dar consejos y ejecutar acciones en beneficio del usuario y luego aprender de la respuesta que reciba por la ejecución de las mismas.
- El agente puede recibir instrucciones directamente del usuario.
- Un agente puede aprender recibiendo consejos desde otros agentes y aprender de las experiencias de estos.

Los agentes de Interfaz trabajan únicamente con el fin de beneficiar a su usuario, pero interactúan con otros agentes para adquirir conocimiento, de esta manera es posible que las habilidades que algunos han adquirido y que les han permitido lograr sus objetivos sean transmitidas a otros para facilitarles las cosas.

1.3.6 PRINCIPALES ARQUITECTURAS DE AGENTES.

- Arquitecturas deliberativas.
- Arquitecturas reactivas.
- Arquitecturas hibridas.
- Arquitecturas basadas en comportamientos.
- Arquitecturas de razonamiento práctico y BDI.

1.3.6.1 ARQUITECTURAS DELIBERATIVAS.

Son basadas en los enfoques tradicionales propuestos por la inteligencia artificial simbólica. Su filosofía es *“pensar mucho, luego actuar”*. Se fundamentan en la

hipótesis, el comportamiento inteligente puede ser logrado mediante un sistema de símbolos que permite la composición y procesamiento de símbolos que representan entidades en el mundo. Las decisiones y conclusiones son obtenidas mediante razonamiento lógico automatizado.

Algunas características son:

- Existe una representación interna central (RIC) con una descripción (simbólica) del estado actual del ambiente, las acciones del agente, y sus objetivos.
- La información sensorial es fusionada y traducida en una descripción simbólica que es utilizada para actualizar el estado actual del ambiente.
- El razonamiento sobre las acciones a realizar involucra desarrollar una planeación que determina un curso de acción para alcanzar un objetivo.
- La ejecución de los planes es mayoritariamente de lazo abierto.

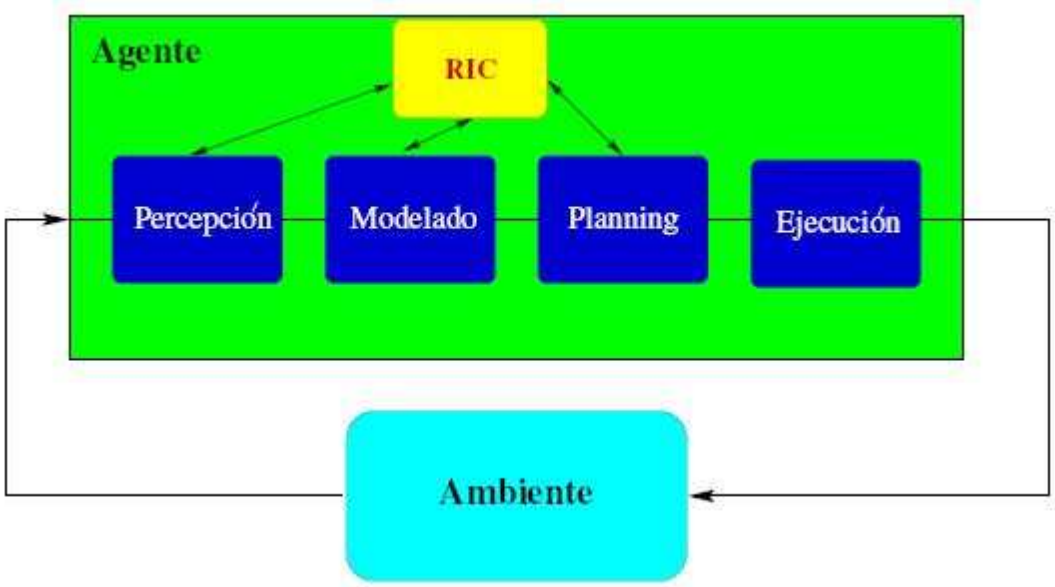


Ilustración 7 ARQUITECTURA DELIBERATIVA TIPICA.

1.3.6.2 ARQUITECTURAS REACTIVAS.

Surgen a partir de críticas sobre la aplicabilidad de la inteligencia artificial simbólica tradicional a problemas de robótica. Su filosofía es *"no pensar, actuar"*, se fundamentan en la hipótesis que para construir un sistema inteligente es necesario tener las representaciones situadas en el mundo físico.

Existe un fuerte acoplamiento entre las percepciones sensoriales y las acciones motoras para permitir que el agente responda rápidamente a ambientes cambiantes y poco estructurados. Son inspiradas en modelos biológicos de estimulo-respuesta y aplicadas principalmente en robótica. No existe ningún tipo de estado interno, ni se realiza ningún tipo de razonamiento. El comportamiento inteligente emerge de la interacción entre el agente y su ambiente y entre varios comportamientos más simples.

Algunas características son:

- El sistema es descompuesto en módulos de competencia o comportamientos orientados a una tarea específica dependiente del problema, ejemplo: evitar obstáculos, aproximarse a una puerta.
- Cada modulo es conectado directamente a sus sensores y efectores relevantes.
- La comunicación entre los módulos se realiza por medio de mensajes elementales, energía de activación o simples señales de inhibición y supresión.
- Los módulos también se suelen comunicar.
- El desarrollo de los agentes es incremental.
- La estructura de control es descentralizada y altamente distribuida.
- No existen objetivos ni planes explícitos.
- La actividad hacia un objetivo es una propiedad emergente de distintos tipos de interacciones.
- El rol de interacciones es fundamental entre módulos, módulos con el ambiente y entre agentes.

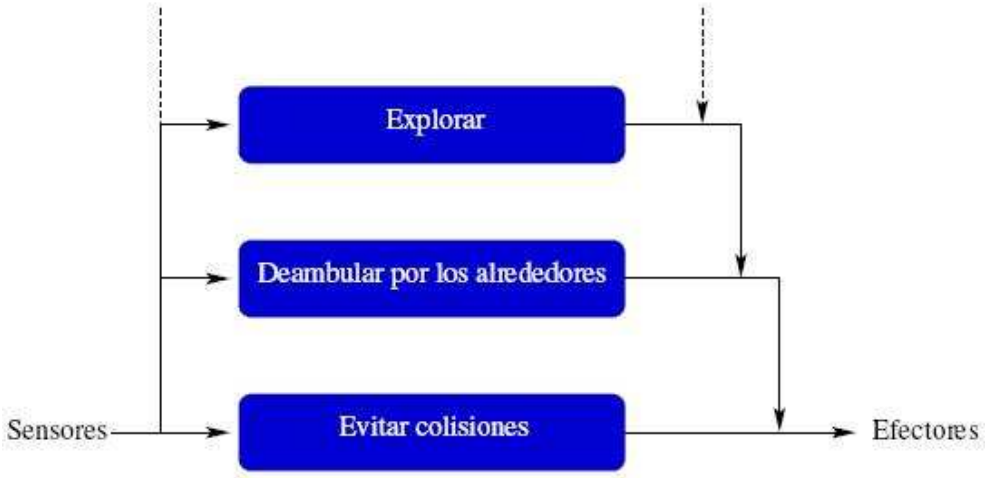


Ilustración 8 ARQUITECTURA REACTIVA TIPICA.

1.3.6.3 ARQUITECTURAS HIBRIDAS.

Combinan las respuestas de tiempo real de las arquitecturas reactivas, con la racionalidad de las deliberativas. Su filosofía puede ser resumida como *"pensar y actuar independientemente, en paralelo".* Se organiza en capas:

Capa Reactiva: rápida reacción, comportamientos primitivos. Ningún o mínimo estado interno.

Capa Intermedia: Ejecución condicional de planes. Estado reflejando memoria sobre el pasado.

Capa Deliberativa: Elaboración de planes. Estado refleja predicciones sobre el futuro.

Algunas características son:

- Componente reactiva: responde a las necesidades inmediatas del agente y opera a escalas de tiempos cortos.
- Componente deliberativa: usa representaciones (simbólicas) internas del mundo altamente abstracto y opera sobre ellas a escala de tiempo más largas.

- Componente intermediaria: Coordina el funcionamiento de las componentes previas. La componente reactiva debe dominar a la deliberativa cuando se presentan cambios inesperados y críticos en el ambiente. La componente deliberativa debe informar a la reactiva para guiar al agente hacia estrategias más eficientes y optimas.

Ilustración 9 ARQUITECTURA HIBRIDA TIPICA.

1.3.6.4 ARQUITECTURAS BASADAS EN COMPORTAMIENTOS.

Surgen del enfoque reactivo, compensando sus limitaciones (falta de estado, incapacidad de considerar el pasado y futuro) y conservan sus fortalezas (respuestas de tiempo real, escalabilidad y robustez).

Algunas características son:

- División de capas similar a las hibridas, pero las capas no difieren significativamente en términos de escala tiempo.
- Cada capa es un comportamiento dependiente de una tarea específica.
- Varios comportamientos pueden activarse simultáneamente ante la entrada sensorial.

- Los comportamientos pueden tener objetivos diferentes y posiblemente incompatibles (conflicto de objetivos).
- Coordina salidas de múltiples comportamientos para realizar la acción más satisfactoria mediante ciertos mecanismos.

Los mecanismos para seleccionar la acción a ejecutar (mecanismos de coordinación de comportamientos) son:

- Mecanismos Competitivos: Varios comportamientos compiten simultáneamente para tomar el control del agente. Solo uno, o un subconjunto de ellos es habilitado para participar.
- Mecanismos Cooperativos: Permiten que múltiples comportamientos contribuyan al control final del agente.
    - Mecanismos indirectos: La salida de cada comportamiento contiene información acerca de las acciones, para determinar la acción a ser seleccionada.
    - Mecanismos de fusión de comandos: La salida de cada comportamiento es combinada directamente con la salida de los comportamientos restantes, con el objetivo de construir la acción que será enviada a los efectores.

1.3.6.5 ARQUITECTURAS DE RAZONAMIENTO PRÁCTICO Y BDI.

Términos como “creencia”, “deseo” o “intención”, proceden del modelo BDI (Belief-Desire-intention). Este modelo se basa parcialmente en las ideas del filósofo estadounidense *Daniel Clement Dennett*, quien defiende el enfoque intencional para explicar el comportamiento humano. Según Dennett el comportamiento humano puede explicarse e incluso predecirse basándose en conceptos de intenciones como querer, desear, temer, entre otros, él utiliza el término sistema intencional para referirse a entidades “cuyo comportamiento puede predecirse por el método de atribuir creencias, deseos y perspicacia racional”.

El modelo BDI también se basa en la teoría de razonamiento práctico, propuesta por el filósofo estadounidense *Michael Bratman*. Esta teoría define un marco sicológico de sentido común para entender el comportamiento humano mediante creencias, deseos e intenciones, concebidos como planes parciales para realizar acciones concretas. A diferencia del razonamiento puramente lógico, el razonamiento practico se orienta hacia las acciones (¿Qué debemos hacer para conseguir algo?). Es innegable que el razonamiento humano es práctico, pues casi todas las situaciones con la que nos enfrentamos en la vida ordinaria requieren soluciones prácticas, pues nadie en un viaje utilizaría las ecuaciones de Euler-Lagrange para calcular el camino más corto entre dos ciudades. Según Bratman, el razonamiento práctico se basa en:

- Decidir qué objetivos deben conseguirse (proceso de decisión o elección)
- Elegir un plan para conseguirlos (planning).

En el modelo BDI, los agentes inteligentes se consideran sistemas intencionales. Es decir, su comportamiento se explica considerándolos agentes racionales cuyas acciones están determinadas por sus deseos y creencias.

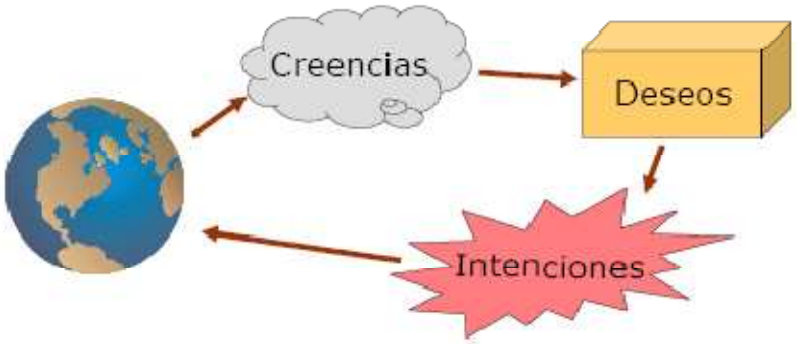


Ilustración 10 ARQUITECTURA BDI

En el modelo BDI, las *creencias* representan el conocimiento que el agente tiene sobre el estado actual del entorno, de sí mismo y de otros agentes. Se utiliza el término

creencia en lugar de conocimiento porque las creencias de un agente pueden ser erróneas. Todos los agentes inteligentes como los humanos toman, para cumplir sus objetivos, decisiones basadas en sus creencias sobre el mundo.

Los humanos y los agentes tienen una gran flexibilidad en sus creencias, si descubren que son erróneas o incompletas pueden cambiarlas, también disponen de reglas de inferencia que les permita revisar sus creencias cuando se encuentran en situaciones que contradicen las anteriores. Cabe destacar que las creencias humanas son mucho más complejas que la de los agentes, estas se codifican en nuestras redes neuronales y dependiendo de estímulos externos las neuronas pueden ser más complejas. En los agentes inteligentes, las creencias se codifican mediante *lenguajes lógicos* y *ontologías*. Por ejemplo, la ausencia de una persona puede codificarse como la creencia (Carlos estaEnCasa ausente), donde ausente es una constante que significa que la persona no está en casa o no esta a la vista del agente y Carlos es una instancia de la clase Persona. Usar lenguajes lógicos y ontologías limita la riqueza expresiva de las creencias.

Los *deseos*, también son llamados metas u objetivos, representan determinados estados que el agente quiere producir en su entorno, en sí mismo o en otros agentes. Corresponden a tareas que el agente debe realizar como: “ganar a las cartas”.

Las *intenciones*, son deseos que el agente se compromete a realizar (no todos los deseos pueden cumplirse), Son deseos muy limitados a los aspectos del entorno sobre los cuales el agente tiene algún grado de control. Un agente solo tiene intenciones que considera posibles y que cree que puede lograr actuando de cierta manera.

La influencia de la teoría de Bratman sobre el razonamiento práctico en el modelo BDI, lleva a las siguientes conclusiones:

- Las salidas del proceso de decisión son las intenciones.
- Las salidas del proceso de búsqueda de medios para alcanzar su objetivo son los planes, que se ejecutan mediante acciones, y estas provocan que se puedan cambiar las creencias, deseos, intenciones, planes y acciones del agente.

- Si las acciones del plan no le permiten conseguir su intención, intentará otro plan.

1.4 SISTEMAS MULTIAGENTE (SMA).

Consiste en un grupo de agentes que interactúan, generalmente a través de mensajes sobre una red de computadores. También puede definirse como una red débilmente acoplada de solucionadores de problemas (agentes) que trabajan conjuntamente para resolver problemas que superan las capacidades individuales o conocimiento de cada uno de ellos.

Ilustración 11 DESCRIPCION DE SISTEMAS MULTIAGENTE.

Algunas características son:

- Existe más de un agente en el sistema.
- Los agentes son autónomos, distribuidos, posiblemente heterogéneos y pueden ser individualistas o cooperativos.
- Cada agente tiene información incompleta, o capacidades limitadas para resolver un problema, por lo que cada agente tiene un punto de vista limitado.
- Un SMA (sistema multi-agente), es usualmente abierto (P2P) y no existe un diseño centralizado.
- No existe un control global del sistema.

- Los datos están descentralizados.
- La computación es asíncrona.
- Se provee de alguna infraestructura que especifica los *protocolos de comunicación e interacción*.

El creciente interés en la investigación de los SMA está motivado por la posibilidad de proveer robustez a partir de la redundancia y eficiencia a partir del paralelismo y la habilidad para resolver problemas en los que los datos, el control o habilidades están distribuidos.

En los SMA, pueden existir diferentes tipos de agentes, entre los que se encuentran, los agentes colaborativos, de interface, móviles y de información.

1.4.1 PROTOCOLOS DE COMUNICACIÓN E INTERACCION.

El diseñador de un SMA puede imponer un protocolo de interacción y una estrategia para cada agente. Existe una biblioteca FIPA que contiene los siguientes protocolos de interacción:

PROTOCOLOS	DESCRIPCION
Request	Un agente pide que otro realice cierta acción.
Query Ip	Un agente pide que otro realice cierta acción sobre otro agente.
Request When	Un agente pide que otro realice cierta acción cuando cierta condición lógica sea cierta.

Contract Net	Un agente solicita m propuestas, para lo que envía un call for proposal (cfp) con la tarea y las condiciones sobre la ejecución. Los receptores son potenciales contratistas.
Iterated Contract Net	Análogo a un cfp, pero con la posibilidad de revisar la propuesta.
English Auction	Un agente subasta “a la alta”. Solicita m propuestas enviando un cfp con la tarea y las condiciones sobre la ejecución.
Dutch Auction	Un agente subasta “a la baja” (gana el que menos ofrece). Solicita m propuestas enviando un cfp con la tarea y las condiciones sobre la ejecución.
Brokering	Un agente bróker (intermediario), facilita un conjunto de servicios de comunicación a otros agentes.
Recruiting	Un agente recruiter (reclutador), es un tipo de agente bróker que facilita un conjunto de servicios de comunicación a otros agentes.
Subscribe	Un agente se suscribe a las notificaciones de cambios sobre cierto

	objeto.
Propose	Un agente propone realizar ciertas acciones tan pronto como los receptores lo acepten.

Tabla 2 PROTOCOLOS DE INTERACCION.

Estos protocolos trabajan bajo ciertas primitivas de actos de comunicación, a continuación se mencionará algunas de ellas:

PRIMITIVAS	DESCRIPCION
Accept-proposal	Aceptación de una propuesta para realizar cierta acción. Respuesta de propose.
Agree	Acuerdo para realizar cierta acción. Respuesta de request.
Cancel	Informa a un agente que el emisor no tiene intención de que el segundo efectúe cierta acción.
Call for proposal (cfp)	Pide candidaturas para realizar cierta acción. Inicia procesos de negociación.
Confirm	Informa al receptor de que cierta

	proposición es cierta, siempre que sea cierto que el receptor tiene incertidumbre sobre ella.
Disconfirm	Contrario de confirm.
Failure	Informa que se intento cierta acción pero falló.
Refuse	Acción de rehusar realizar cierta acción, con su explicación.

Tabla 3 PRIMITIVAS DE ACTOS DE COMUNICACION.

1.5 COMPARATIVA ENTRE SMA Y OBJETOS.

Los métodos de producción de software convencionales que parten de un modelo conceptual orientado a objetos están muy arraigados y es poco apreciable las ventajas que los agentes conceden.

AGENTES	OBJETOS
Autonomía de decisión	Ejecuta modelos invocados
Flujo de control propio	Flujo de control llamante
Encapsula la activación del comportamiento	Encapsula el estado y el comportamiento
Estado mental: objetivos, creencias	Estado: valor de las variables

Comportamiento: como decir lo que hacer	Comportamiento: salida a partir de una entrada
Interacciones: actos del habla (intencionalidad)	Mensajes invocan procedimiento
Organización: relaciones sociales	Asociaciones entre objetos

Tabla 4 COMPARATIVA SMA Y OBJETOS.

1.6 COMPARATIVA ENTRE SMA Y SISTEMAS EXPERTOS.

Un sistema experto (SE), es una rama de la Inteligencia Artificial y es aquel que imita las actividades de un humano para resolver problemas de distinto índole. Este se basa en el conocimiento declarativo (hechos sobre objetos, situaciones) y el conocimiento de control (información sobre el seguimiento de una acción), mientras que los agentes evocan unas grandes ventajas.

AGENTES	SISTEMAS EXPERTOS
Interactúan con el entorno	Sistemas cerrados
Distribución de la toma de decisiones: Comportamiento emergente	Sistemas de decisión centralizados
Mayor grado de interacción con el usuario	Interacción con el usuario bajo petición del usuario
Interacción con otros agentes	Normalmente carecen de habilidades sociales

Tabla 5 COMPARATIVA ENTRE SMA Y SISTEMAS EXPERTOS.

CAPITULO II. ADMINISTRACION DE JADE.

2.1 INTRODUCCION.

Este capítulo describe como instalar y ejecutar Jade. Además muestra todas aquellas herramientas para la monitorización de agentes.

2.2 OBTENER JADE.

Es muy fácil, se descarga de la dirección http://jade.tilab.com , después de llenar un breve registro, para cumplir los requisitos de la licencia LGPL. Están disponibles 5 archivos comprimidos:

- El código fuente de Jade.
- El código fuente de los ejemplos.
- La documentación.
- Los archivos binarios de Jade.
- La distribución completa con todos los anteriores archivos, JADE-all-3.6.1, que es la actual versión disponible.

2.3 EJECUCION DE JADE.

Se descomprime el archivo, y se obtienen cuatro archivos, se vuelven a descomprimir esos cuatro archivos en el mismo directorio y se genera un directorio raíz que es *jade* y un subdirectorio *lib* en el que se encuentran las librerías necesarias para la ejecución de jade.

El subdirectorio *lib*, debe ser agregado a la variable de entorno CLASSPATH, por los archivos JAR que contiene.

Después de agregar el subdirectorio al CLASSPATH, se procede a lanzar el contenedor principal de la plataforma, que está compuesto por el agente DF, el agente AMS y el registro RMI, con el siguiente comando:

java jade.Boot [options] [AgentSpecifier list]

Adicionalmente se pueden lanzar más contenedores en el mismo host, o en host remotos.

Un contenedor de agentes puede lanzar usando:

java jade.Boot –container [options] [AgentSpecifier list]

Si no se actualiza la variable CLASSPATH existe otra forma de ejecutar Jade:

java –jar lib\jade.jar –nomtp [options] [AgentSpecifier list]

Donde el parámetro “-nomtp” es para que no se lance la excepción de que no se encuentra la librería http.jar.

2.3.1 OPCIONES DE JADE DESDE LA LINEA DE COMANDOS.

A continuación se describirán cada una de las opciones que se pueden colocar en el comando **java jade.Boot Option* AgentSpecifier***:

- -container: Especifica que la instancia de Jade es un contenedor, y este debe estar ligado a un contenedor principal.
- -host: Especifica el nombre del host donde está corriendo o ejecutando, el contenedor principal. El valor por defecto es localhost.

- -port: Especifica el puerto donde se está corriendo o ejecutando el contenedor principal. El valor por defecto es el puerto 1099.

- -gui: Indica que el RMA (Remote Monitoring Agent), entorno grafico de Jade debe ser lanzado.

- -local-host: Especifica el nombre del host donde va a correr el contenedor actual.

- -local-port: Especifica el número del puerto donde el contenedor actual puede ser contactado. El valor por defecto es el puerto 1099.

- -name: Esta opción es para darle a la plataforma un alias o un nombre simbólico. Lo más recomendable es usarla solo en caso de que se trate de un contenedor principal. Por defecto se genera un identificador con el nombre del host y el número de puerto.

- -container-name: Es para darle un alias o un nombre simbólico al contenedor actual.

- -services: Son los servicios que se cargan y se activan en el arranque. Especificamente estos son una lista de clases. Por defecto están activos los servicios de movilidad y notificación de eventos. Los servicios disponibles en la distribución de Jade 3.6 son los siguientes:

NOMBRE DEL SERVICIO	CLASE A LA QUE PERTENECE	ASPECTOS A RESALTAR
Messaging	jade.core.messaging.MessagingService	ACL intercambio de mensajes y Administración MTP. Activación automática

Agent-Management	jade.core.management.AgentManagementService	Administración básica del ciclo de vida del agente. Activado automaticamente
Agent-Mobility	jade.core.mobility.AgentMobilityService	Habilita la movilidad. Activado por defecto.
Notification	jade.core.event.NotificationService	Es requerido para el agente sniffer y el introspector. Activado por defecto.
Persistent-Delivery	jade.core.messaging.PersistentDeliveryService	Manejo de la cantidad de mensajes ACL. Servicio inactivo.
Main-Replication	jade.core.replication.MainReplicationService	Es usado para prueba de fallos haciendo

		replicas del contenedor principal. Inactivo.
Address-Notification	jade.core.replication.AddressNotificationService	Notificaciones cuando haya algún cambio en la lista de los contenedores principales activos. Inactivo.
UDPNodeMonitoring	jade.core.nodeMonitoring.UDPNodeMonitoringService	Es usado para monitorizar los paquetes UDP entre los nodos. Inactivo.

Tabla 6 SERVICIOS DE JADE.

- -mtp: Especifica una lista de Message Transport Protocols para ser activados el contenedor actual.

- -nomtp: Tiene más precedencia que –mtp y lo sobrescribe. Se usa para sobrescribir el comportamiento del contenedor principal que usa por defecto.

- -backupmain: Indica que la instancia de Jade actual es un backup o una copia del contenedor principal. Este debe ser ligado con el contenedor principal.

- -smhost: Esta opción es usada cuando una copia del contenedor principal es iniciada, y es para seleccionar el nombre del host donde está el Service Manager o administrador de servicios, que generalmente es localhost. Esta opción se usa en los nodos donde el servicio de Main-Replication está activo.

- -smport: Es para seleccionar el puerto TCP[6] donde está el Service Manager o administrador de servicios, cuando una copia del contenedor principal es iniciada. Esta opción se usa en los nodos donde el servicio de Main-Replication está activo.

- -smaddrs: Con esta opción, se puede listar todas las copias del contenedor principal, de esta manera un contenedor periférico, se puede conectar a alguna copia. Esta opción es una alternativa a la de usar la activación del servicio Address-Notification.

- -aclcodec: Todos los mensajes son codificados por String-Based ACLCodec. Con esta opción se puede especificar una lista adicional de códigos ACLCodec para que los agentes decodifiquen y codifiquen mensajes utilizando estos codigos.

[6] TCP: Transmission Control Protocol.

- -nomobility: Deshabilita la movilidad y la clonación en el contenedor que se encuentra en ejecución. A pesar de esto el contenedor aun tiene la capacidad de lanzar agentes. Esta opción esta deshabilitada por defecto.

- -version: muestra la versión de Jade.

- -help: muestra la ayuda de Jade.

- -conf: Se usa para cargar o guardar toda la configuración de Jade. Se puede especificar también desde archivo, si este se coloca como argumento.

2.3.2 SINTAXIS DESDE LA LINEA DE COMANDOS.

Para mostrar la sintaxis utilizada en la línea de comandos se usará una representación conocida como Extended Backus-Naur Form[7] (EBNF), con reglas comunes y las definiciones de tokens.

java jade.Boot Option* AgentSpecifier*

Option = "-container"

| "-backupmain"

| "-gui"

| "-nomtp"

| "-services" ClassName (";" ClassName)*

| "-host" HostName

[7] EBNF: es una meta sintaxis usada para expresar gramáticas libres de contexto, una manera formal de describir lenguajes formales.

```
                | "-port" PortNumber
                | "-local-host" HostName
                | "-local-port" PortNumber
                | "-name" PlatformName
                | "-container-name" ContainerName
                | "-smaddrs" HostName (":" PortNumber )? (";" HostName
                (":" PortNumber )? )*
                | "-mtp" ClassName "(" Argument* ")"
                (";" ClassName "(" Argument* ")")*
                | "-aclcodec" ClassName (";" ClassName )*
                | "-nomobility"
                | "-version"
                | "-help"
                | "-conf" FileName?
                | "-" Keyword Value
ClassName = PackageName? Word
PackageName = (Word ".")+
Argument = Word | Number | String
HostName = Word ( "." Word )*
```

PortNumber = Number

AgentSpecifier = AgentName ":" ClassName ("(" Argument* ")")?

AgentName = Word

PlatformName = Word

Keyword = Word

Value = Word

2.3.3 EJECUCION DE AGENTES DESDE LA LINEA DE COMANDOS.

Los agentes pueden ser lanzados desde la línea de comandos, como se puede notar en la parte final del comando general **[AgentSpecifier list]**. Esto se puede hacer de la siguiente manera:

Primero se coloca el nombre del agente y luego el nombre de la clase Java que lo implementa, separados por dos puntos ‘: ’, se le puede pasar los argumentos a la clase colocándolos entre paréntesis.

Por ejemplo: *Peter:myAgent* esto significa “crear un nuevo agente llamado Peter del cual su implementación es un objeto de la clase myagent”.

Otro ejemplo: *Peter:myAgent(“today is raining” 123*) esto significa “crear un nuevo agente llamado Peter del cual su implementación es un objeto de la clase myAgent y se le pasa un arreglo dos argumentos a su constructor, el primero es today is raining y el segundo es 123”.

Ahora veamos un ejemplo bastante ilustrativo y completo, de cuál es la manera correcta de ejecutar agentes desde la línea de comandos.

Primero se actualiza la variable de entorno CLASSPATH en Windows 9x/NT se hace con el siguiente comando:

set CLASSPATH=%CLASSPATH%;.;c:\jade\lib\jade.jar;

c:\jade\lib\jadeTools.jar;c:\jade\lib\Commons-codec\commonscodec-

1.3.jar;c:\jade\lib\iiop.jar

Se ejecuta el contenedor principal de la plataforma. Supongamos que el nombre del host de esta máquina es “kim.cselt.it”.

prompt> java jade.Boot –gui

Luego, se ejecuta el siguiente comando para iniciar un contenedor de agentes en otra máquina, indicándole que se una a la plataforma que se está corriendo en el host “kim.cselt.it” e iniciar un agente:

prompt> java jade.Boot -host kim.cselt.it -container

sender1:examples.receivers.AgentSender

Donde “sender1” es el nombre del nuevo agente y examples.receivers.AgentSender es el código fuente donde se encuentra implementado.

Por último en una tercera maquina se inicia otro contenedor de agentes, indicándole que se una a la plataforma que se está corriendo en el host “kim.cselt.it” y se inician dos nuevos agentes, esto se hace de la siguiente manera:

prompt> java jade.Boot –host kim.cselt.it –container

receiver2:examples.receivers.AgentReceiver

sender2:examples.receivers.AgentSender

Donde el agente que se llama “sender2”, esta implementado por la clase examples.receivers.AgentSender, y “receiver2” esta implementado por la clase examples.receivers.AgentReceiver.

2.3.4 CONFIGURACION DEL SERVICIO DE PAGINAS AMARILLAS.

El agente DF, es quien brinda el servicio de páginas amarillas, este agente tiene diferentes configuraciones opcionales que pueden ser entradas desde la línea de comandos o con un archivo plano.

PARAMETROS OPCIONALES	DESCRIPCION
jade_domain_df_autocleanup	Inmediatamente cuando algún agente termina con su actividad, el agente DF se encarga de limpiar el registro o quitarlo del registro. Por defecto esta en false.
jade_domain_df_maxleasetime	Indica el tiempo máximo en milisegundos, en que el agente DF permite el registro de agentes. Por defecto es infinito.
jade_domain_df_maxresult	Es el número máximo de ítems que pueden ser encontrados en una operación de búsqueda en las páginas amarillas. Por defecto es 100.
jade_domain_df_kb-factory	Con este parámetro se puede especificar el nombre de una clase

	llamada factory class, que es usada por el agente DF para manejar toda la información que se va acumulando en los catálogos. Por defecto es jade.domain.DFKBFactory.
jade_domain_df_db-default	Si se coloca en true, esto quiere indicar que el DF guardara el catalogo en una base de datos HSQLDB. Esta es la mejor forma de manejar la persistencia de datos del catalogo. Para usar este parámetro las clases java de HSQLDB deben ser agregadas al CLASSPATH. HSQL se puede descargar de http://sourceforge.net/projects/hsqldb.
jade_domain_df_db-url	Define la URL de JDBC donde se va a guardar el catalogo. Con este parámetro el agente DF se puede configurar para manejar otras bases de datos diferentes a HSQLDB.
jade_domain_df_db-cleantables	Si se coloca en true, borra todo el contenido de las tablas en la base de datos usadas por el DF. Este parámetro se ignora cuando el catalogo no se guarda en una base de datos, sino en memoria. Por defecto es false.
jade_domain_df_db-driver	Indica el controlador JDBC que se usa para accesar a la base de datos. Por

	defecto es ODBC-JDBC bridge.
jade_domain_df_db-username	Indica el nombre de usuario para accesar a la base de datos. Por defecto esta en null.
jade_domain_df_db-password	Indica la contraseña para accesar a la base de datos. Por defecto esta en null.

Tabla 7 OPCIONES PARA EL MANEJO DE LAS PÁGINAS AMARILLAS.

Ejemplos:

Para ejecutar un contenedor principal con el agente DF guardando el catalogo en una base de datos HSQLDB se usa el siguiente comando:

java jade.Boot -jade_domain_df_db-default

Previamente se tuvo que definir correctamente el CLASSPATH con las librerías de HSQLDB.

Para ejecutar un contenedor principal que guarde el catalogo en una base de datos accesible desde una URL jdbc:odbc:dfdb y se mantenga registrando agentes por al menos 1 hora, se usa el siguiente comando:

java jade.Boot -jade_domain_df_db-url jdbc:odbc:dfdb

-jade_domain_df_maxleasetime 3600000

El siguiente comando ejecuta un contenedor de Jade con un agente DF que se llama “df1” y lee su configuración desde un archivo llamado df1conf.properties.

java jade.Boot –container –host <main-container-host>

df1:jade.domain.df(df1conf.properties)

El contenido del archivo df1conf.properties se muestra a continuación:

```
#DF configuration file

# DF Catalogue stored in a database (URL is jdbc:odbc:dfdb)
jade_domain_df_db-url = jdbc:odbc:dfdb

# Maximum granted lease time for agent registrations: 1 hour
jade_domain_df_maxleasetime = 3600000
```

Ilustración 12 EJEMPLO DE UN ARCHIVO DE CONFIGURACION PARA DF.

2.3.5 IDENTIFICADORES DE AGENTES.

Para realizar comunicaciones entre agentes es necesario que estos estén identificados de forma única en la plataforma. De acuerdo con las especificaciones de FIPA, cada agente posee un Identificador de Agente (AID = Agent Identifier). La estructura del AID se compone de varios campos, siendo los más importante los de nombre y dirección. El nombre del agente es un identificador global único para él. JADE construye dicho identificador concatenando el nombre dado al agente en la plataforma por el usuario con el nombre de la plataforma inicial del agente (HAP = Home Agent Platform) separa dos por el carácter '@'. Un ejemplo de este nombre sería peter@kim:1099/JADE, donde "peter" es el nombre del agente y "kim:1099/JADE"el de la plataforma. Sólo los nombres completos son válidos dentro de los mensajes ACL. Las direcciones por el contrario pueden contener varias direcciones de transporte en las cuales contactar al agente. La sintaxis de estas direcciones es una secuencia de URL.

2.4 MANEJO Y MONITORIZACION DE AGENTES EN LA INTERFAZ GRAFICA.

Se han construido algunas herramientas gráficas de Jade, para facilitar tareas como la depuración de aplicaciones multi-agente. Cada herramienta es un agente y obedece las mismas reglas que un agente común.

2.4.1 AGENTE RMA.

El agente de monitoreo remoto (Remote Monitoring Agent) controla el ciclo de vida de una plataforma de agentes y de todos los agentes registrados. Además como en su nombre lo indica, este habilita el control remoto, donde la interfaz gráfica es utilizada para controlar la ejecución de agentes y su ciclo de vida.

Un agente RMA es un objeto de Java, que se instancia de la clase jade.tools.rma.rma, y este puede ser lanzado o ejecutado desde la línea de comandos como cualquier agente ordinario, de la siguiente manera:

java jade.Boot myConsole:jade.tools.rma.rma, o también **java jade.Boot –gui**

En la misma plataforma pueden ejecutarse más de un agente RMA, diferenciándolos con el nombre local que tengan, pero en un contenedor solo se puede ejecutar uno.

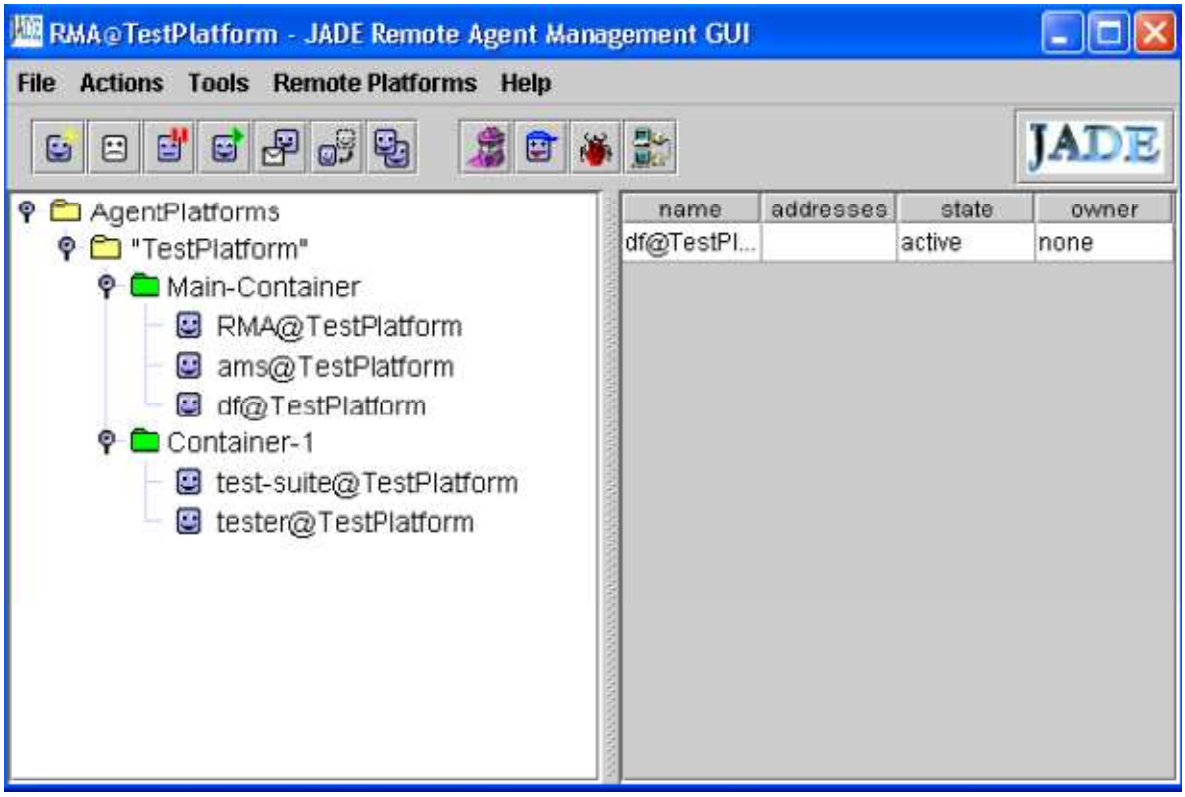


Ilustración 13 CAPTURA DE LA INTERFAZ DE UN AGENTE RMA.

A continuación se mostraran los comandos que pueden ser ejecutados desde la barra de menú del agente RMA.

- File menú: Contiene los comandos de uso general de RMA.
    - Close RMA Agent: invoca el método doDelete() finalizando el agente RMA. Tiene el mismo efecto que cerrar la ventana.
    - Exit this container: finaliza el contenedor donde se está ejecutando el agente RMA y todos los agentes que estén corriendo en él. Si el contenedor es el contenedor principal toda la plataforma será finalizada.
    - Shut down Agent Platform: se sale de la plataforma, finalizando todos los contenedores y los agentes.

- Actions menú: son todas las acciones administrativas que se necesitan para usar la plataforma.
    - Start New Agent: crea un nuevo agente. Al usuario se le sugiere un nombre y la clase Java de la que será instanciado el nuevo agente. Se puede seleccionar el contenedor donde se quiere que sea lanzado.
    - Kill Selected Items: esta acción elimina todos los agentes y los contenedores seleccionados y los borra del registro de la plataforma.
    - Suspend Selected Agents: esta acción deja inactivo a cualquier agente que este seleccionado, equivale a llamar el método doSuspend().
    - Resume Selected Agents: es equivalente a llamar el método doActivate(). Cuando un agente está suspendido este lo coloca en el estado AP_ACTIVE.
    - Send Custom Message to Selected Agents: Al seleccionar esta acción se despliega un cuadro de dialogo donde se puede enviar un mensaje ACL a un agente.

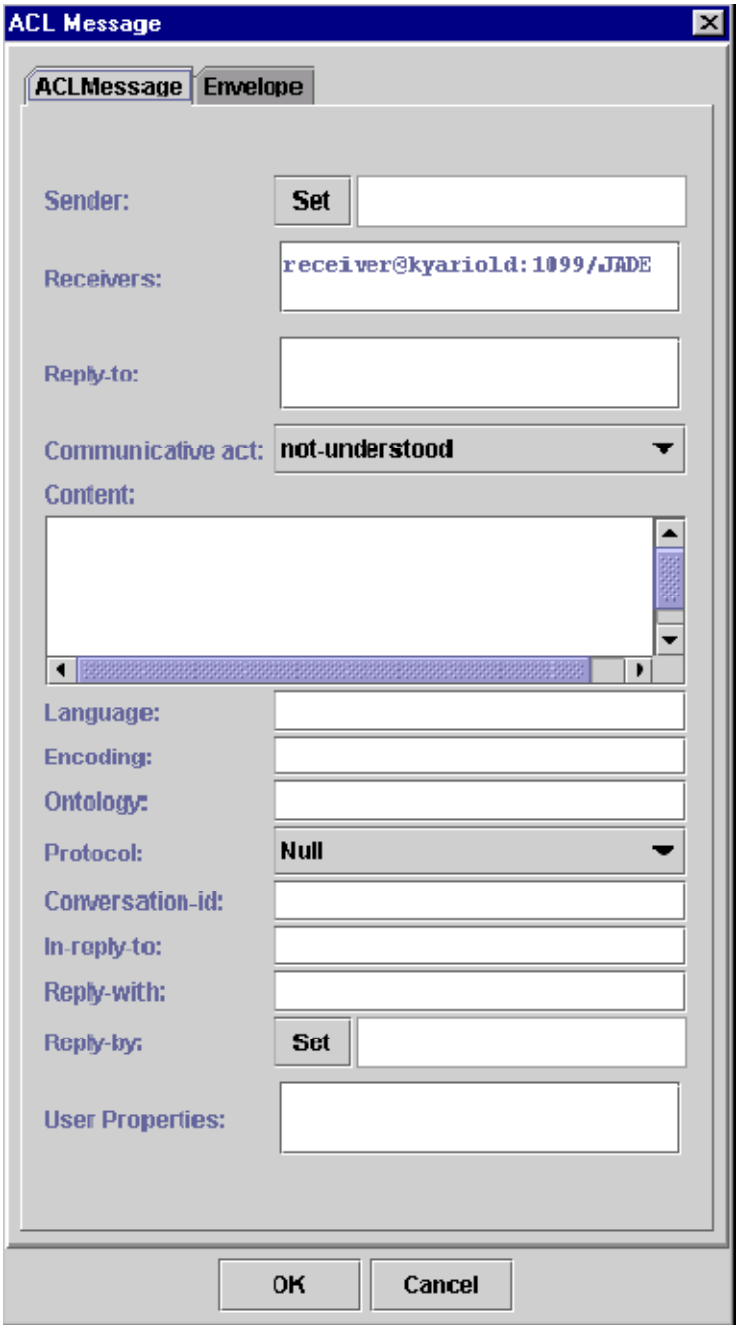


Ilustración 14 CUADRO DE DIALOGO PARA ENVIAR MENSAJES ACL.

- Migrate Agent: migra agentes de un contenedor a otro, pero no todos los agentes pueden migrar, esto depende de su implementación.
- Clone Agent: clona el agente que este seleccionado, se le debe indicar el nuevo nombre y el contenedor donde el agente se va a ejecutar.

- Tools menú: Este menú contiene todos los comandos para iniciar las herramientas que provee Jade.

- RemotePlatforms menú: Este menú habilita el control de algunas plataformas remotas que sigan las especificaciones FIPA. Estas plataformas no necesariamente tienen que ser de Jade.
    - Add Remote Platform vía AMS AID: esta acción obtiene la descripción de una plataforma de agentes remota a través de AMS. Se requiere insertar la información del AID del AMS remoto, para que la plataforma remota se pueda agregar al árbol que se muestra en la interfaz del RMA.
    - Add Remote Platform vía URL: agrega la descripción de una plataforma remota vía URL. Se debe insertar la URL para que la plataforma remota se muestre en el árbol de la interfaz gráfica del agente RMA.
    - View APDescription: es para ver la descripción de una plataforma seleccionada.
    - Refresh APDescription: esta acción llama a través del AMS remoto la descripción de una plataforma y hace una recarga de la misma.
    - Remove Remote Platform: elimina la plataforma remota que esta seleccionada en la interfaz gráfica.
    - Refresh Agent List: hace una búsqueda en la plataforma remota de todos los agentes que estén contenidos en ella, mostrándolos en forma de árbol en la interfaz del agente RMA.

2.4.2 AGENTE DUMMY.

El agente Dummy es una herramienta que permite a los usuarios componer y enviar mensajes ACL, manteniendo una lista de todos los mensajes ACL enviados y recibidos. Cada mensaje de la lista puede ser visto o modificado. Este agente puede ser ejecutado cuantas veces sea necesario. Existen dos maneras de ejecutar este agente, a través de la línea de comandos, o a través del menú de herramientas del agente RMA. El comando es el siguiente:

Java jade.Boot theDummy:jade.tools.DummyAgent.DummyAgent

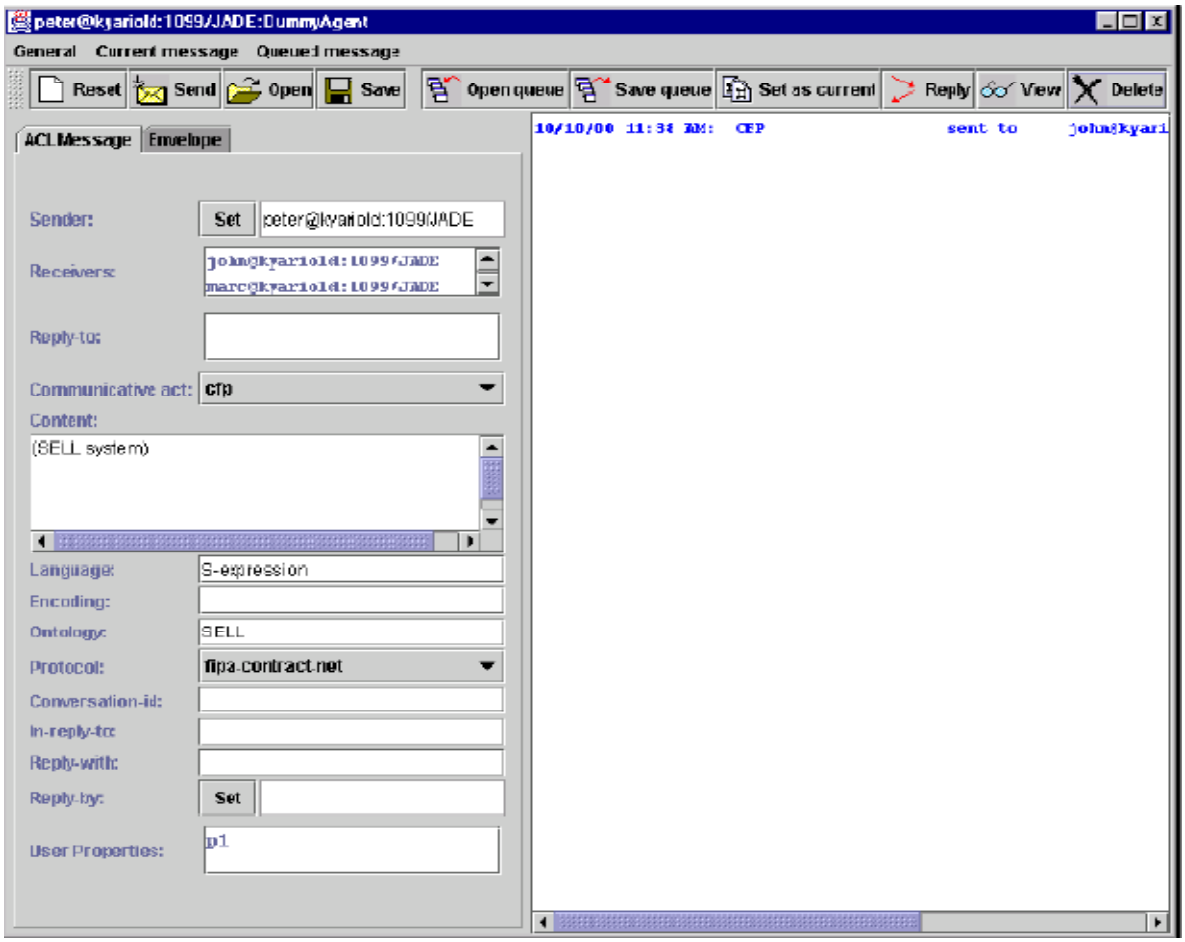


Ilustración 15 CAPTURA DE LA INTERFAZ DE UN AGENTE DUMMY.

2.4.3 AGENTE DF INTERFAZ GRAFICA.

Este agente se ejecuta desde el menú de herramientas en la interfaz del RMA, la interfaz gráfica del agente DF solo puede ser mostrada en el host donde está corriendo la plataforma (contenedor principal).

La interfaz gráfica del agente DF se usa para interactuar con el DF, es decir, se puede ver las descripciones de todos los agentes registrados, así mismo, registrar y quitar registros de agentes, modificar las descripciones de los agentes y buscar algún agente a través de su descripción.

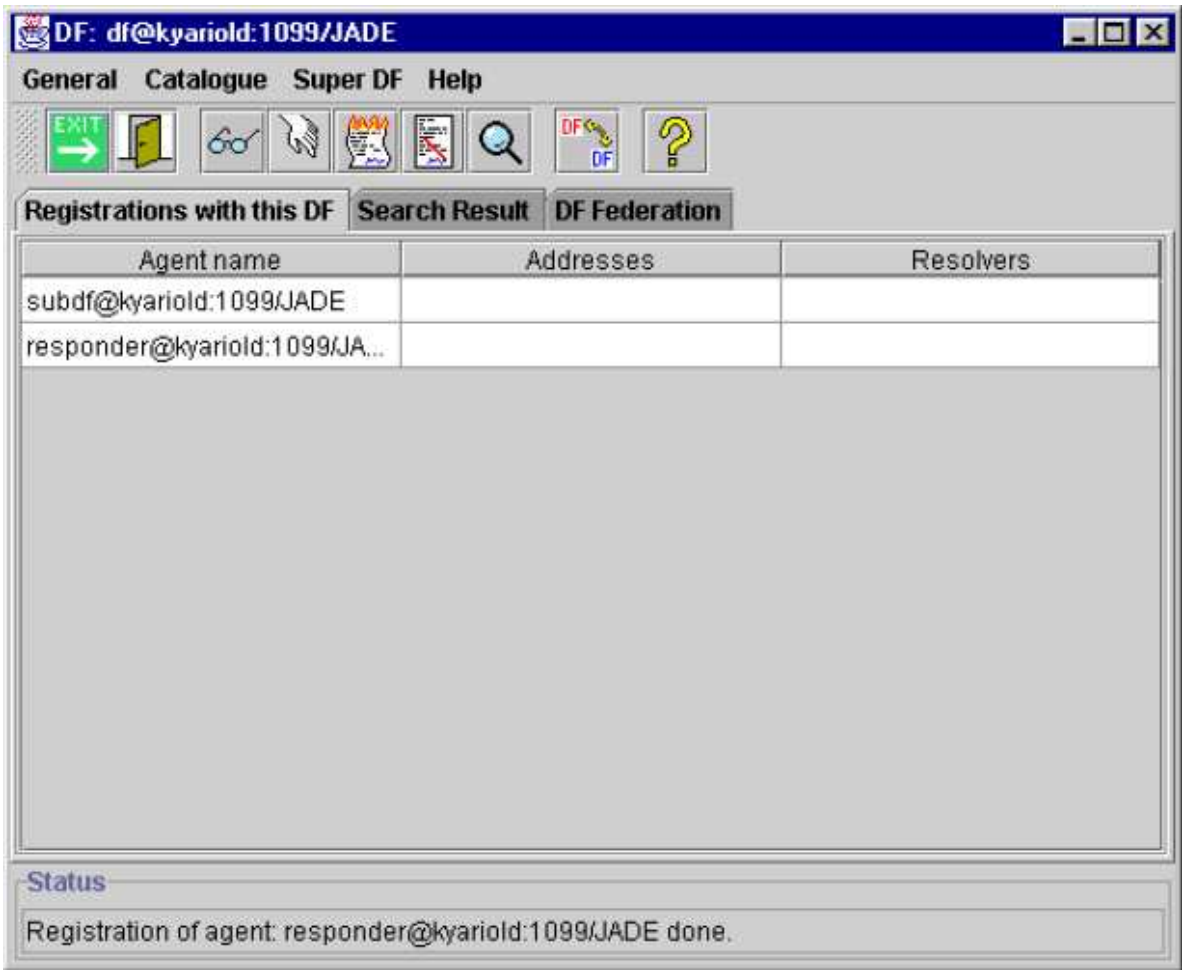


Ilustración 16 CAPTURA DE LA INTERFAZ GRAFICA DEL AGENTE DF.

2.4.4 AGENTE SNIFFER.

Existen dos maneras de ejecutar este agente, a través de la línea de comandos, o a través del menú de herramientas del agente RMA. El comando es el siguiente:

java jade.Boot sniffer:jade.tools.sniffer.Sniffer

Este agente es el que se encarga de interceptar los mensajes ACL lanzados y los visualiza gráficamente en un modo similar a los diagramas de secuencia de UML. Resulta útil para depurar conversaciones entre agentes. El usuario puede ver todos los mensajes de un agente o un grupo de agentes y guardarlos en el disco.

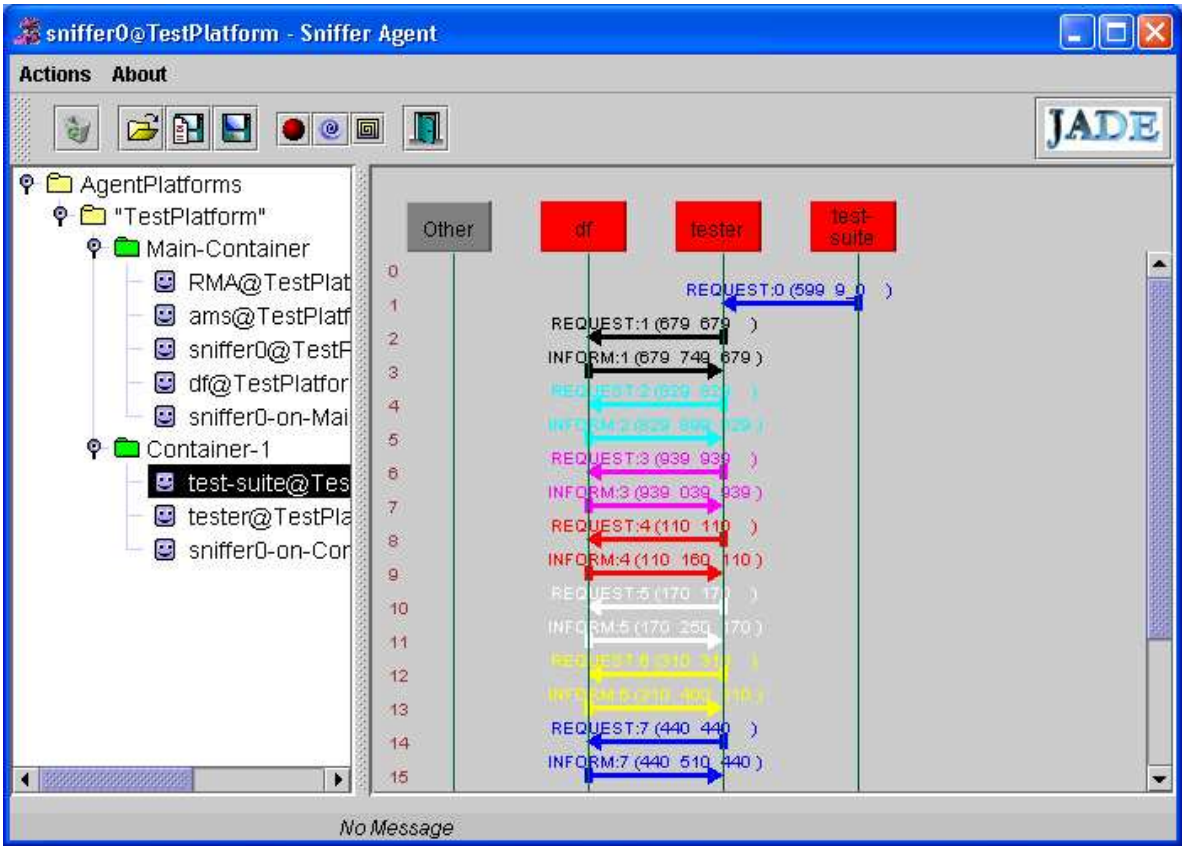


Ilustración 17 CAPTURA DE LA INTERFAZ GRAFICA DE UN AGENTE SNIFFER.

El agente Sniffer al iniciar se inscribe con la plataforma, entonces, este informa cada vez que un agente nace o muere, de igual manera cuando un contenedor es creado o es eliminado.

2.4.5 AGENTE INTROSPECTOR.

Este agente permite monitorizar el ciclo de vida de los agentes y los mensajes ACL que intercambian. Los agentes son pasados como argumentos a través de la línea de comandos, o a través de un archivo de configuración.

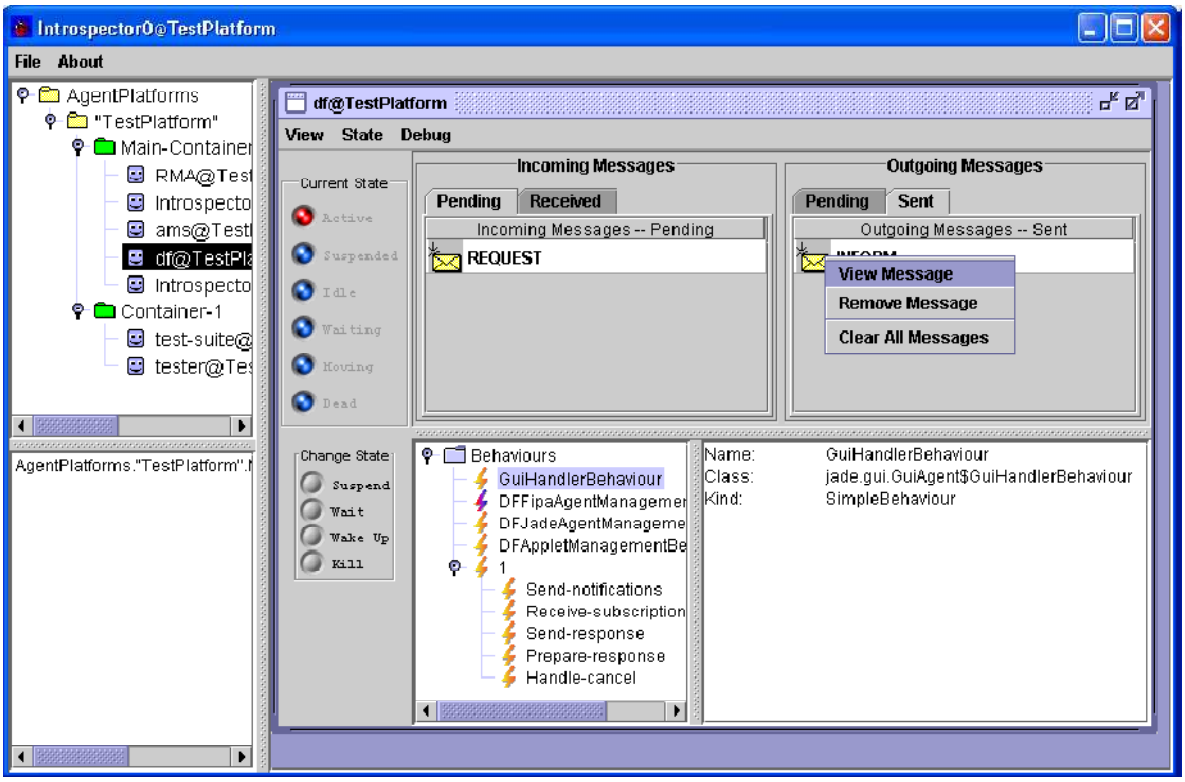


Ilustración 18 CAPTURA DE LA INTERFAZ GRAFICA DEL AGENTE INTROSPECTOR.

CAPITULO III. PROGRAMACION EN JADE.

3.1 INTRODUCCION.

En este capítulo se presentan conceptos básicos, que son necesarios conocer antes de adentrarse en la programación de sistemas multi-agente.

Para empezar en la programación de agentes se debe tener en cuenta que Jade es apto para la integración con herramientas de desarrollo como NET-BEANS O ECLIPSE, solo basta con asociar todos los .jar de jade con un proyecto.

También da una idea general acerca de cómo empezar a programar un agente en Jade. Y muestra un ejemplo muy claro y de mucha utilidad como lo es el programar el agente Hola Mundo.

3.2 AGENTES.

Para la programación de agentes de Jade, se debe definir una clase que representa al agente que debe heredar de la clase jade.core.agent, y en ella se tienen que implementar todos aquellos comportamientos que queremos que el agente manifieste.

Los agentes poseen nombres únicos en el entorno de ejecución, y estos son implementados como un único hilo de ejecución (simple.threaded).

Manejan métodos que indican inicio y fin, estos métodos son el método protegido setup(), que inicializa al agente incluyendo las instrucciones que especificarán la ontología a utilizar y los comportamientos asociados al agente, y el método protegido takeDown(), que se encarga de liberar los recursos antes de la eliminación del agente, este método es invocado cuando se realiza una llamada al método doDelete(), que es el que finaliza la ejecución de un agente. Los comportamientos que se describen en la clase se utilizan básicamente para el envío y recepción de mensajes.

3.3 COMPORTAMIENTOS.

Un comportamiento o behaviour hace referencia a una funcionalidad que incorpora el agente. Los comportamientos especifican tareas o servicios que realiza un agente para lograr sus objetivos. Cada comportamiento puede realizar una tarea simple o compuesta dependiendo de la implementación.

Los agentes están programados sobre la base de sus comportamientos. La programación basada en comportamientos debe realizar los siguientes pasos:

- Determinar qué debe ser capaz de hacer el agente.
- Asociar cada funcionalidad con un comportamiento.
- Escoger el tipo de comportamientos.
- Dejar a Jade la tarea del scheduling o planificación (un solo comportamiento se está ejecutando en cada instante).

Los comportamientos o behaviours tienen dos métodos principales:

- action(): que es el método que se ejecuta cada vez que se invoca el método
- done(): hace la comprobación de si el comportamiento ha terminado o no.

3.3.1 TIPOS DE COMPORTAMIENTOS.

Dentro de Jade, vienen implementado diferentes comportamientos listos para ser usados. Son los siguientes:

- Clase SimpleBehaviour: Representa un comportamiento atómico.
- Clase OneShotBehaviour: Representa un comportamiento que se debe ejecutar solo una vez, por eso, su método done() retorna true, si no es redefinido.
- Clase CyclicBehaviour: Representa un comportamiento que debe ejecutarse una serie de veces. El método done, si no es redefinido, devuelve false.

- Clase CompositeBehaviour: Esta clase se compone de diferentes sub-behaviours que se pueden ejecutar siguiendo diferentes políticas de planificación. Las diferentes políticas vienen determinadas por la subclase elegida, SequentialBehaviour, ParallelBehaviour y FSMBehavior.
- Clase SequentialBehaviour: Esta clase deriva de CompositeBehaviour y ejecuta subbehaviours de forma secuencial, y termina cuando todos los métodos action() han terminado.
- Clase ParallelBehaviour: Esta clase deriva de CompositeBehaviour y ejecuta los subbehaviours de manera concurrente. En el constructor de la clase se puede especificar cuando se desea que acabe la ejecución:
- Clase FSMBehaviour: Esta clase permite definir una Máquina de Estados Finita mediante sub-behaviours. Cada sub-comportamiento representa un estado de la máquina, y las transiciones se van produciendo según la salida de dichos estados. La finalización se alcanza cuando se termine de ejecutar algún sub-behaviour que se haya registrado como estado final.
- Clase SenderBehaviour: Encapsula la acción de envío de un mensaje ACL. Este mensaje se le debe especificar en el constructor.
- Clase ReceiverBehaviour: Encapsula la acción de recepción de un mensaje ACL. Termina cuando se recibe el mensaje o cuando pasa una cierta cantidad de tiempo especificada en el constructor.
- Clase WakerBehaviour: Implementa un comportamiento honesto que se ejecuta justo después de que se haya pasado un tiempo especificado.

3.4 LENGUAJE DE COMUNICACIÓN DE AGENTES ACL.

La comunicación entre agentes es fundamental para que un sistema multi-agente funcione. Por medio de estos mensajes se determina el comportamiento social de los agentes.

Para que los agentes se puedan comunicar deben usar el mismo lenguaje de comunicación, y se debe definir cuál es el tipo de mensaje, es decir, si es para informar, solicitar, preguntar, entre otros tipos de mensajes.

Las conversaciones entre agentes se rigen por una serie de protocolos de interacción, que en su mayoría se muestran en la tabla 2.

El lenguaje de comunicación de agentes (ACL) permitirá transmitir una serie de conocimiento que vendrá expresado en un lenguaje de contenido. Los términos del lenguaje de contenido que representen conocimiento pertenecerán a un vocabulario común a los distintos agentes que se llama ontología.

3.5 ONTOLOGIAS.

Los agentes, para comunicarse, necesitan compartir una representación común del conocimiento. Esta representación debe incluir todo lo que los agentes necesiten decirse durante la comunicación. La ontología se puede ver como el idioma que usan los agentes para hablar. Si dos agentes no hablan el mismo idioma no se entenderán.

Jade proporciona tres formas distintas de llevar a cabo la comunicación entre agentes:

La forma más básica es utilizar cadenas para representar el contenido de los mensajes.

Otra forma es utilizar objetos serializables[8] de Java, que transmitirían directamente el contenido de los mensajes. Este es el método más conveniente para aplicaciones locales donde todos los agentes son implementados en Java. Un inconveniente es que estos mensajes no son entendibles para las personas.

El tercer método consistiría en definir los objetos que van a ser transferidos como extensión de las clases predefinidas por Jade que pueden codificar/decodificar los

[8] La serialización es el proceso de convertir el estado de un objeto a un formato que se pueda almacenar o transportar.

mensajes a un formato FIPA estándar. Esto permite que los agentes de Jade puedan inter operar con otros sistemas de agentes.

Una ontología en Jade, se define de forma que los agentes se comuniquen utilizando el tercer método descrito. El soporte Jade para ontologías incluye las clases para trabajar con estas y con los lenguajes de contenido:

Los lenguajes de contenido tienen que ver con la representación interna del contenido de los mensajes ACL.

Las ontologías tienen que ver con la semántica de los mensajes que se intercambian y su chequeo.

3.6 MOVILIDAD.

La movilidad de un agente es la habilidad para que este migre o haga una copia de sí mismo (clonarse) a través de uno o múltiples nodos de una red. El soporte de movilidad en Jade consiste en un conjunto de clases y métodos que permiten a un agente, ejecutar las acciones requeridas por sí mismo o por el AMS (Agent Management System), y una ontología específica de movilidad (MobilityOntology). Cada instancia de ejecución en Jade se llama contenedor. El conjunto de todos los contenedores se llama plataforma y proporciona una capa homogénea que esconde los agentes, la complejidad y diversas plataformas. La versión actual de Jade sólo soporta movilidad intra-plataforma, esto es, un agente solo puede moverse dentro de la misma plataforma de contenedor en contenedor. Esto quiere decir que:

- Se involucran varios hosts.
- Cada host tiene su contenedor.
- La migración puede suceder a petición del propio agente.

3.7 ESQUELETO DE UN PROGRAMA EN JADE.

El agente esqueleto no es más que un ejemplo de agente sencillo que se puede usar como esqueleto para desarrollar nuestros propios agentes, es una base para poder implementar nuestro código, ya que no se le ha asociado ningún comportamiento ni ninguna ontología.

```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.DFService;
import jade.domain.FIPAException;
public class Esqueleto extends Agent {
   protected void setup() {

        /** Registrarse con el agente DF, páginas amarillas */
        DFAgentDescription dfd = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType("AgenteEsqueleto");
        sd.setName(getName());
        sd.setOwnership("DCCIA");
        dfd.setName(getAID());
        dfd.addServices(sd);
        try {
           DFService.register(this,dfd);
        } catch (FIPAException e) {
           System.err.println(getLocalName()+" registro en DF no realizado. Razon: "+e.getMessage());
           doDelete();
        }
   }
}//end class Esqueleto
```

Ilustración 19 AGENTE ESQUELETO.

El agente esqueleto es una subclase de la clase Agent, por esta razón se pueden crear agentes de la clase Esqueleto utilizando java jade.Boot. En el método setup se incluyen las instrucciones adecuadas para registrar el agente en el Directory Facilitator (DF).

El agente Esqueleto no hace ninguna tarea, por tal razón se considera un agente sin utilidad. Lo que diferencia un agente de otro son las tareas que este pueda desarrollar.

La creación de un comportamiento consiste en crear una clase privada en el interior de la clase del agente en cuestión, y asociarle esa clase mediante el método addBehaviour. Los comportamientos no se ejecutan concurrentemente, es decir que, es necesario que cada comportamiento libere el control para permitir que el resto de comportamientos se ejecuten.

3.8 EJEMPLO HOLA MUNDO.

Este es un reconocido ejemplo típico para cualquier lenguaje de programación, que se caracteriza por su sencillez.

Generalmente este es un ejemplo que es útil como prueba de configuración, para asegurar que Jade está instalado correctamente y funcionando.

```
import jade.core.Agent;
import jade.core.behaviours.*;


public class Simple0 extends Agent
{
    protected void setup()
    {
        addBehaviour( new B1( this ) );
    }
}

class B1 extends SimpleBehaviour
{
    public B1(Agent a) {
         super(a);
    }

    public void action()
    {
       System.out.println( "Hello World! My name is " +
              myAgent.getLocalName() );
    }

    private boolean finished = false;
    public  boolean done() {  return finished;  }

} //End class B1
```

Ilustración 20 HOLA MUNDO EN JADE.

Como podemos notar del ejemplo las acciones de los agentes son especificadas en clases behaviours, más exactamente en el método action de la clase.

La clase B1 es una subclase de SimpleBehaviour, la cual es la más común de las predefiniciones para los comportamientos de Jade.

Después de cada acción o behaviour del agente, hay que darle fin, para esto es el método done que es una bandera que indica si la acción terminó o no.

En la variable local myAgent se almacena la referencia pasada como parámetro cuando el comportamiento es creado, es decir en este caso es el nombre.

CONCLUSIONES

Esta monografía está desarrollada como manual para mostrar todo el potencial que pueden tener los sistemas desarrollados en Jade. Cabe destacar que durante el desarrollo de esta monografía, he pensado en lo que podría ser un nuevo paradigma de programación, el paradigma de agentes y sistemas multi-agente, que a decir verdad no es tan nuevo, ya hace mucho se han venido haciendo estudios acerca de ello y actualmente constituye un área de creciente interés dentro de la inteligencia artificial.

Los agentes inteligentes tienen el potencial para automatizar una gran cantidad de actividades de la vida diaria, lo cual permitiría a las personas usar más tiempo en analizar resultados y tomar decisiones basados en estos.

Un agente se basa en la racionalidad para la resolución de problemas y la toma de decisiones, siempre dependiendo del ambiente en que este se desenvuelva.

En un sistema multi-agente, el grupo de agentes que lo integran debe trabajar de manera cooperativa e individual. Los agentes trabajan de manera cooperativa para satisfacer las metas globales que se derivan de la búsqueda de las soluciones a los problemas globales y de manera individual, por que las metas globales son descompuestas en submetas, generando metas locales para los agentes que participaran en el desarrollo de las soluciones a los problemas.

Para que dos o más agentes puedan comunicarse es necesario que se entiendan, tanto en el aspecto sintáctico como semántico en los mensajes que se intercambien. Al momento de enviar un mensaje por parte de un agente se podría establecer un símil con el proceso de comunicación humana, es decir, existe un agente emisor, el mensaje como tal, un canal de comunicación, el agente receptor y la interpretación que este le dé al mensaje.

Además, Jade es de fácil instalación y utilización, asimismo proporciona una serie de herramientas que simplifican la administración y desarrollo de una aplicación, como son RMA, Dummy Agent, Snnifer Agent, DF GUI e introspector Agent.

Si tratamos de ver las cosas desde el punto de vista de un diseñador de software, este solo tiene que centrarse, en determinar las características particulares de los agentes, como son los objetivos, las tareas a realizar, para la aplicación especifica que está desarrollando, es decir, la definición de los comportamientos y de las ontologías especificas de la aplicación, donde se engloben todos los conceptos, acciones, predicados y proposiciones que definen el dominio de la aplicación.

Por consiguiente, Jade reduce drásticamente el esfuerzo que supone el diseño y desarrollo de un sistema multi-agente.

BIBLIOGRAFIA

- Belén Díaz Agudo, Rubén Fuentes Fernández, “Introducción a Jade”, Departamento de Sistemas Informáticos y Programación, UCM.
  http://www.scribd.com/doc/2064096/IntroJade

- Gonzalo A. Aranda Corral, “Seminario de Jade”, Sevilla, 23 de Mayo de 2005.
  www.cs.us.es/~garanda/**seminario**/Semin_**Jade**.pdf

- Mario Luis Errecalde, Agentes y Sistemas Multiagente, Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC).
  agentes.unsl.googlepages.com/teo5ag.pdf
  agentes.**unsl**.googlepages.com/teorías

- Jean Vaucher and Ambroise Ncho, “JADE Tutorial and Primer”, septiembre de 2003.
  http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html

- Dominic Greenwood, FIPA The Foundation for Intelligent, Physical Agents.
  http://jade.tilab.com/papers/JADETutorialIEEE/JADETutorial_FIPA.pdf

- César Llamas Bello, “Protocolos de interacción entre agentes”, Doctorado en informática, Sistemas Multiagente, 2002.
  www.infor.uva.es/~cllamas/MAS/ProInterAg.pdf

- Pablo Saud Pérez, TUTORIAL DE JADE.
  http://www.dccia.ua.es/~pablo/tutorial_agentes/index.html

- Paola Neri Ortiz, La Plataforma JADE para desarrollo de Agentes.
  http://www.geocities.com/paolaneriortiz/jade.htm

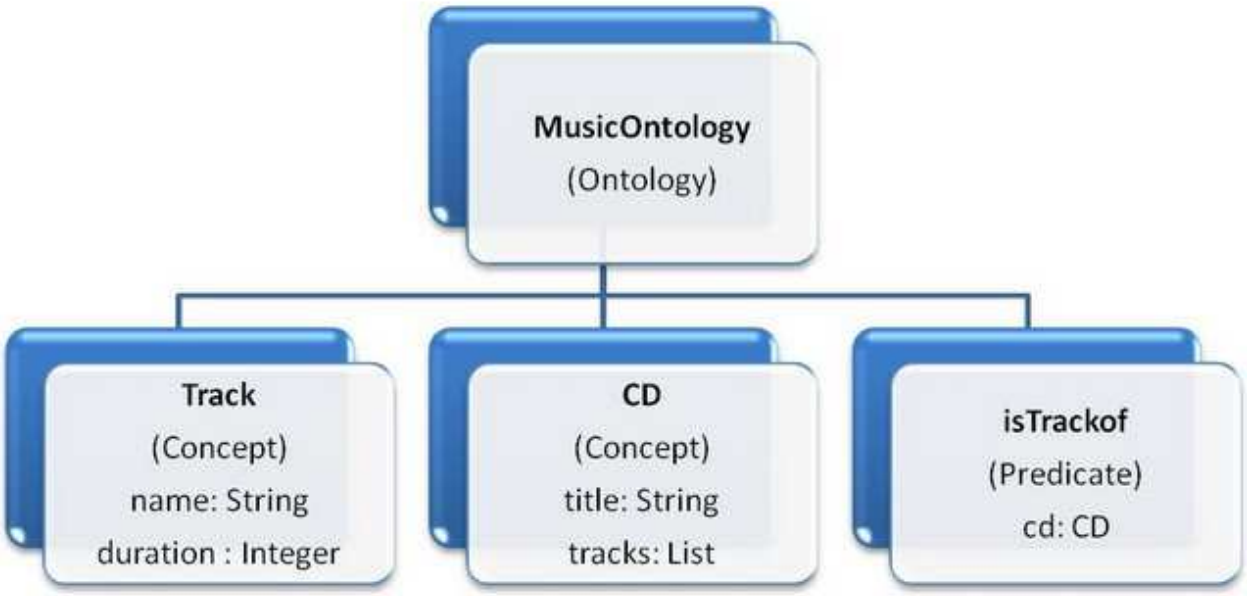
- Escuela Superior de Ingeniería Informática de la Universidad de Vigo, Programación en JADE.
  http://programacionjade.wikispaces.com/

- Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa, JADE PROGRAMMER´S GUIDE.
  http://jade.tilab.com/doc/programmersguide.pdf

ANEXO A

EJEMPLO MusicOntology

El ejemplo consiste en el intercambio de información musical entre dos agentes y se encuentra disponible en http://programacionjade.wikispaces.com/Ontologías. Para enviar la información relativa a un disco utilizamos la ontología definida MusicOntology. La estructura de esta ontología se refleja en la siguiente gráfica:



Las clases que se deben crear son las siguientes:

- MusicOntology.java: La definición de la ontología denominada musicOntology, la cual extiende a jade.content.onto.Ontology.
- Track.java: La definición del concepto Track con los atributos name y duration, que son de tipo predefinido String e Integer respectivamente. Esta clase debe implementar la interfaz Concept.(jade.content.Concept)

- CD.java: La definición del concepto CD con los atributos title y tracks, de tipo String y List respectivamente. También debe implementar la interfaz Concept.
- isTrackof.java: La definición del predicado isTrackof con el atributo CD, el cual implementa la interfaz Predicate (jade.content.Predicate).

Ambos conceptos (Track y CD) serán instanciados como ConceptSchema y el predicado isTrackof como PredicateSchema en la definición de la ontología en el fichero MusicOntology.java. Estas cuatro clases conforman el paquete musicOntology que deberá ser importado por las clases que empleen la ontología.

Para empezar se deberá definir el vocabulario, en él incluimos los nombres de atributos y clases que forman los conceptos, predicados y acciones. Los términos serán declarados como constantes.

Código de **MusicOntology.java**:

```
package musicOntology;
import jade.content.onto.*;
import jade.content.schema.*;
public class MusicOntology extends Ontology {
    // Definimos el vocabulario
  public static final String ONTOLOGY_NAME = "Music-ontology";
  public static final String CD = "CD";
  public static final String CD_TITLE = "title";
  public static final String CD_TRACKS = "tracks";

  public static final String TRACK = "TRACK";
  public static final String TRACK_NAME = "name";
  public static final String TRACK_DURATION = "duration";
  public static final String IS_TRACK_OF = "isTrackof";
    public static final String TRACKS = "Tracks";

  // Obtenemos la instancia de la ontología
    private static Ontology theInstance = new MusicOntology();

    public static Ontology getInstance() {
       return theInstance;
    }
```

```
  //Constructor
  private MusicOntology() {
     super(ONTOLOGY_NAME,  BasicOntology.getInstance());

    try {
    //asociamos las clases de los conceptos
        add(new ConceptSchema(CD), CD.class);
        add(new ConceptSchema(TRACK), Track.class);
        add(new AgentActionSchema(IS_TRACK_OF), isTrackof.class);

         AggregateSchema tracksSchema = new
AggregateSchema(BasicOntology.SEQUENCE);

         ConceptSchema cdSchema = (ConceptSchema) getSchema(CD);
         cdSchema.add(CD_TITLE, (PrimitiveSchema)
getSchema(BasicOntology.STRING));
         cdSchema.add(CD_TRACKS, tracksSchema);

         ConceptSchema trackSchema = (ConceptSchema) getSchema(TRACK);
         trackSchema.add(TRACK_NAME, (TermSchema)
getSchema(BasicOntology.STRING));
         trackSchema.add(TRACK_DURATION,
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.OPTIONAL);

         PredicateSchema isTrackOfSchema = new PredicateSchema(IS_TRACK_OF);
         isTrackOfSchema.add(CD, cdSchema );

    }
    catch (OntologyException oe) {
        oe.printStackTrace();
    }
    }
}
```

El agente *RecibirCD* debe inicialmente registrar la ontología y el lenguaje. Está compuesto por un solo comportamiento *ReceiverBehaviour* que espera por un mensaje Inform que utilize la ontología *MusicOntology.* Una vez recibido este mensaje, extrae el contenido y comprueba si es una instancia de la clase *isTrackof*. Si es así, muestra el contenido del objeto CD.

Código de **RecibirCD.java**:

```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;

import jade.content.*;
import jade.content.onto.*;
import jade.content.lang.*;
import jade.content.lang.leap.*;
import jade.util.leap.*;
import jade.content.lang.sl.*;

import musicOntology.*;

public class RecibirCD extends Agent {
   private ContentManager manager     = (ContentManager)getContentManager();
   private Codec          codec       = new LEAPCodec();
   private Codec          codec2      = new SLCodec();
   private Ontology       ontologyCD    = MusicOntology.getInstance();

   class ReceiverBehaviour extends SimpleBehaviour {
   private boolean finished = false;
   public ReceiverBehaviour(Agent a) { super(a); }

   public boolean done() { return finished; }
```

```
public void action() {
    for(int c = 0; c < 2; c++) {
    try {
        System.out.println( "[" + getLocalName() + "] Esperando un mensaje...");
        //Espera por un mensaje
        ACLMessage msg = blockingReceive();

        if (msg!= null) {
            if( msg.getPerformative()==ACLMessage.INFORM){

            ContentElement p = manager.extractContent(msg);
            System.out.println("[" + getLocalName() + "] Recibido un mensaje INFORM: ");
            if(p instanceof isTrackof) {
                isTrackof pred = (isTrackof)p;
                CD disco = pred.getCD();

                System.out.println("Titulo del CD:"+ disco.getTitle());

                List canciones = disco.getTracks();

                //Listado de los datos de las canciones
                    Iterator it = canciones.iterator();
                int i=1;
                while (it.hasNext()){
```

```
                    Track cancion = (Track)it.next();
                   System.out.println("    Nombre cancion "+i+": " + cancion.getName());
                    System.out.print("       Duracion cancion "+i+": ");
                     if(cancion.getDuration()!=null)
                         System.out.print(" " + cancion.getDuration());
            System.out.println();
                         i++;
              }
             }
              else {
                 System.out.println( "[" + getLocalName() + "] El mensaje no se ajusta a lo
esperado: CD");
            }


        }
        else
            System.out.println("[" + getLocalName() + "] Mensaje vacio.");
        }
    } catch(Exception e) { e.printStackTrace(); }
    }
    finished = true;
    //doDelete();
    //System.exit(0);
  }
  }
```

```
    protected void setup() {
        manager.registerLanguage(codec2);
        manager.registerOntology(ontologyCD);

        addBehaviour(new ReceiverBehaviour(this));
    }
}
```

El agente EnviarCD, tras registrar la ontología y el lenguaje, crea dos cedés según la estructura de la ontología. Se creará un mensaje cuya ontología será musicOntology y fijará como contenido del mismo, los predicados isTrackof creados con anterioridad en dos iteraciones.

Código de **EnviarCD.java**:

```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;

import jade.util.leap.List;
import jade.util.leap.ArrayList;

import jade.content.*;
import jade.content.onto.*;
import jade.content.lang.*;
import jade.content.lang.leap.*;
import jade.content.lang.sl.*;
```

```
import musicOntology.*;

public class EnviarCD extends Agent {
    // Para manejar el contenido del mensaje
    private ContentManager manager  = (ContentManager)getContentManager();
    // El agente utiliza el lenguaje ""
    private Codec          codec    = new LEAPCodec();
    private Codec          codec2       = new SLCodec();

    // El agente utiliza la ontologia People ontology
    private Ontology   ontologyCD = MusicOntology.getInstance();

    class SenderBehaviour extends SimpleBehaviour {
    private boolean finished = false;

    public SenderBehaviour(Agent a) { super(a); }

    public boolean done() { return finished; }

    public void action() {
        for(int c = 0; c < 2; c++) {
        try {
            //Crea el mensaje y fija los principales campos del mensaje
            ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
```

```
        AID receiver = new AID("receiver", false);


        msg.setSender(getAID());
        msg.addReceiver(receiver);
        msg.setLanguage(codec2.getName());
        //Fija el campo Ontology
        msg.setOntology(ontologyCD.getName());

        //Creamos un CD que será enviado ajustándose a la ontología
musicShopOntology

        List l = new ArrayList();

        Track t1 = new Track();
        t1.setName("Come together");
        t1.setDuration(180);
        l.add(t1);

        Track t2 = new Track();
        t2.setName("Something");
        t2.setDuration(200);
        l.add(t2);

        CD disco1 = new CD();
```

```
disco1.setTitle("Beatles");
disco1.setTracks(l);

isTrackof pred = new isTrackof(disco1);

//Creamos otro CD

List l2 = new ArrayList();

Track t3 = new Track();
t3.setName("Scar Tissue");
t3.setDuration(180);
l2.add(t3);

Track t4 = new Track();
t4.setName("Otherside");
t4.setDuration(200);
l2.add(t4);

Track t5 = new Track();
t5.setName("Get on top");
t5.setDuration(180);
l2.add(t5);

CD disco2 = new CD();
```

```
        disco2.setTitle("RHCP-Californication");
        disco2.setTracks(l2);

        isTrackof pred2 = new isTrackof(disco2);

        // Fija el contenido del mensaje
        if(c==0){
            manager.fillContent(msg, pred);
            System.out.println( "[" + getLocalName() + "] Creando el mensaje con el
contenido:"+disco1.toString());
        }
        else{
            manager.fillContent(msg, pred2);
            System.out.println( "[" + getLocalName() + "] Creando el mensaje con el
contenido:"+disco2.toString());
        }

        // Envia el mensaje
        System.out.println( "[" + getLocalName() + "] Enviando mensaje...");
        send(msg);
        System.out.println( "[" + getLocalName() + "] Mensaje enviado.");
        }
      catch(Exception e) { e.printStackTrace(); }
      }
      finished = true;
```

```
      // doDelete();
       //System.exit(0);
       }
   }

   protected void setup() {
   manager.registerLanguage(codec2);
   manager.registerOntology(ontologyCD);

   addBehaviour(new SenderBehaviour(this));
   }
}
```

Ahora lanzamos la plataforma Jade: **java jade.Boot –gui,** ejecutamos el agente RecibirCD:

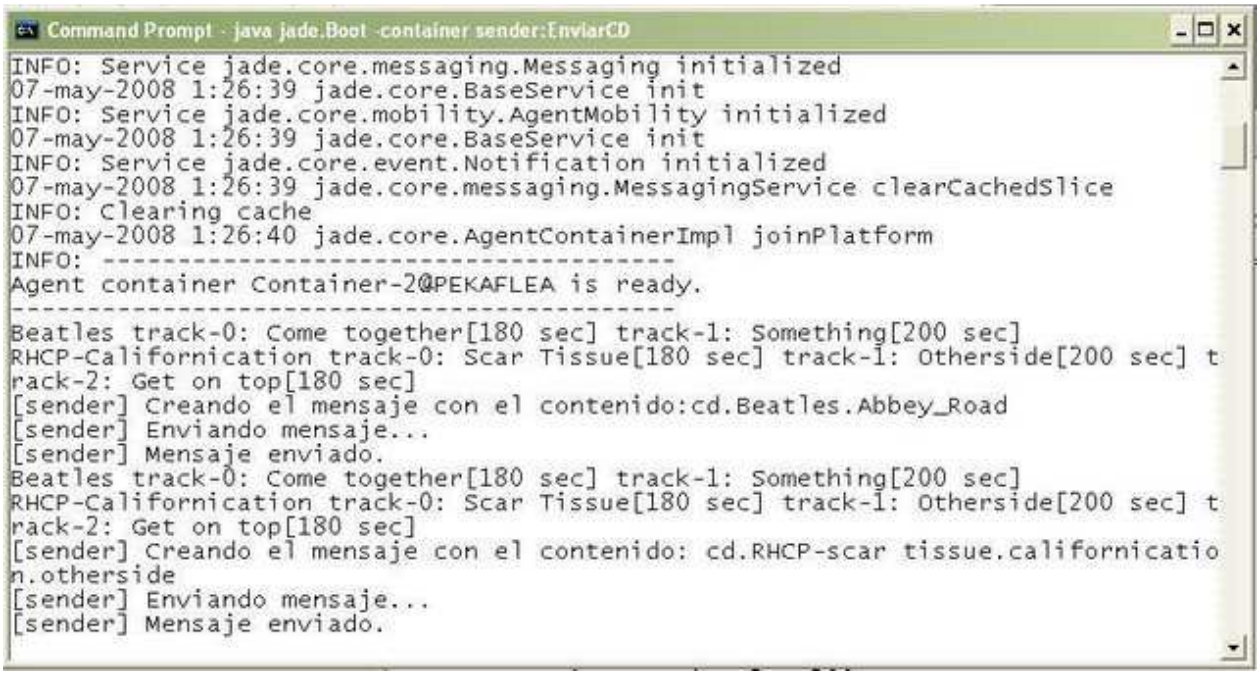
java jade.Boot -container receiver:RecibirCD

Después ejecutamos el agente EnviarCD:

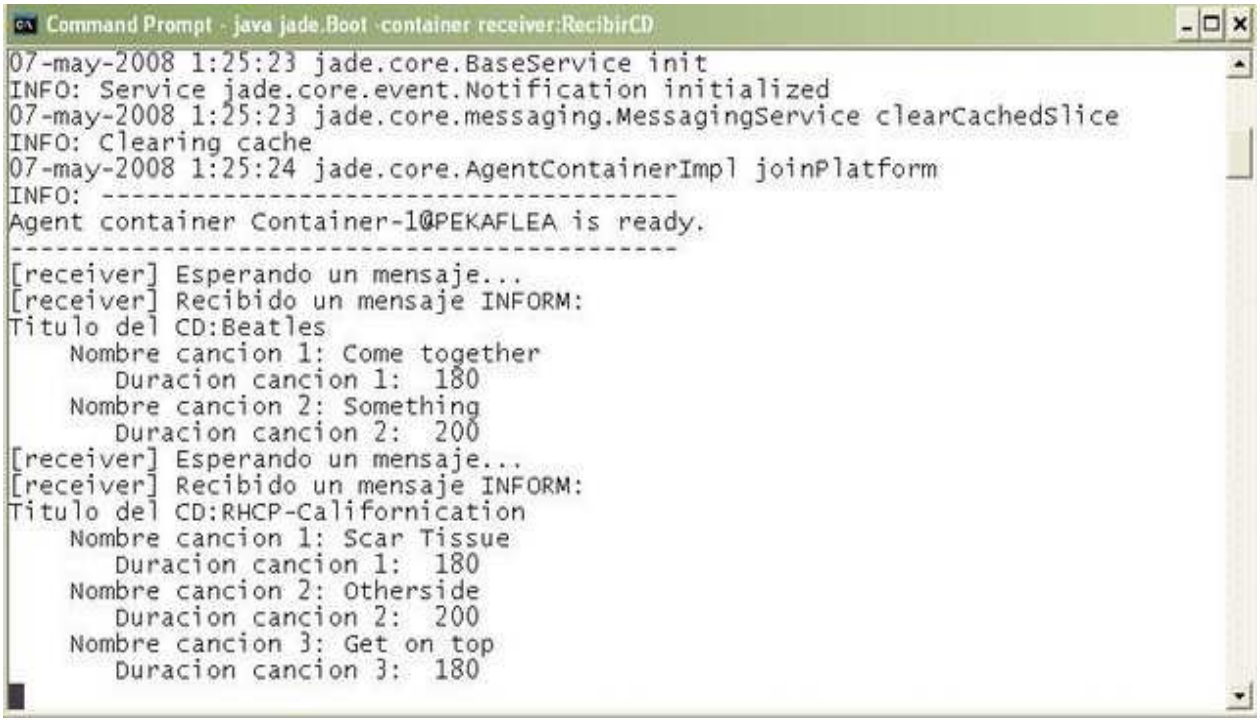
java jade.Boot -container sender:EnviarCD

Finalmente vemos las capturas de la ejecución de ambos agentes:

Captura de **EnviarCD:**



Captura de **RecibirCD:**



ANEXO B

EJEMPLO PingAgent

Este es un ejemplo con el que se puede probar las herramientas gráficas que vienen con Jade. El agente PingAgent está programado de tal forma que contestará a aquellos mensajes con la performativa QUERY-REF y el contenido *ping* que reciba con un mensaje de tipo INFORM. El resto de mensajes que reciba los contestará con mensajes de tipo NOT-UNDERSTOOD.

El agente PingAgent, es un ejemplo que se encuentra en el interior del directorio src/examples, que viene con la descarga que se hace en la página de oficial de Jade.

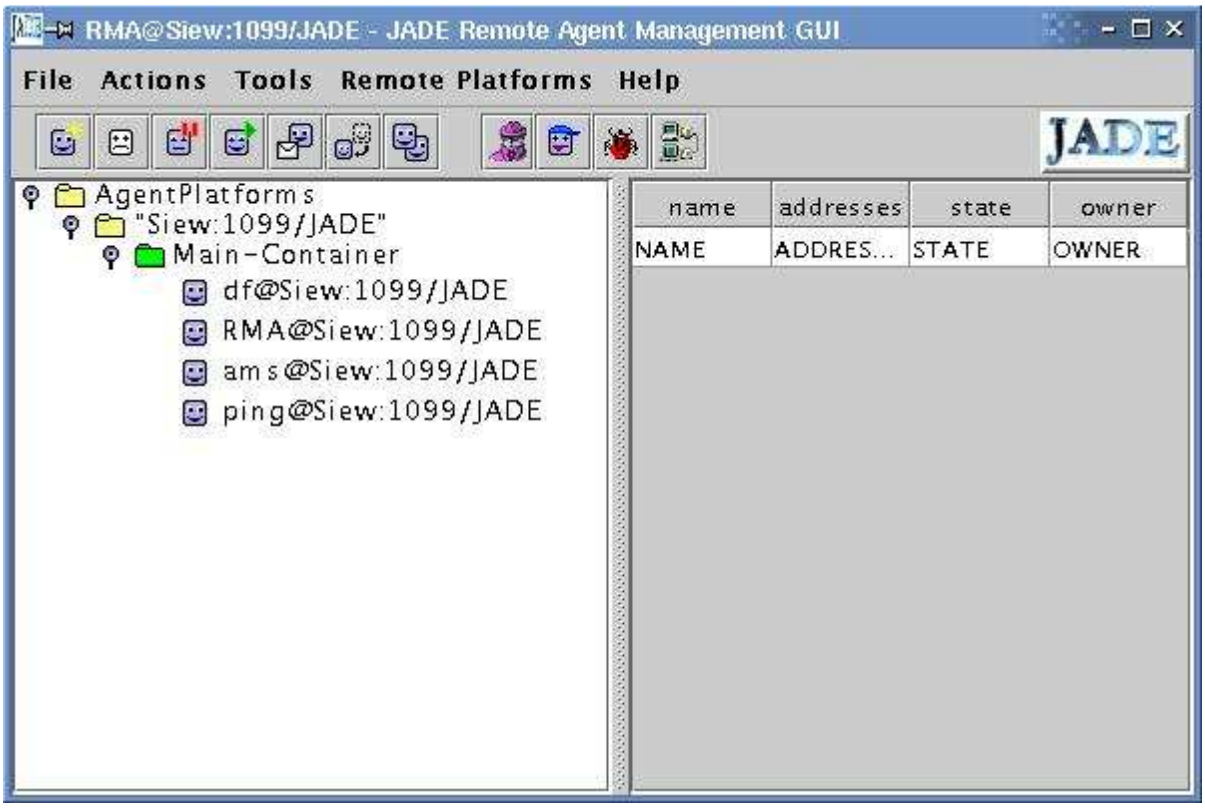
Para compilar este ejemplo, se debe entrar en el directorio de nombre PingAgent y ejecutar el siguiente comando:

javac *.java

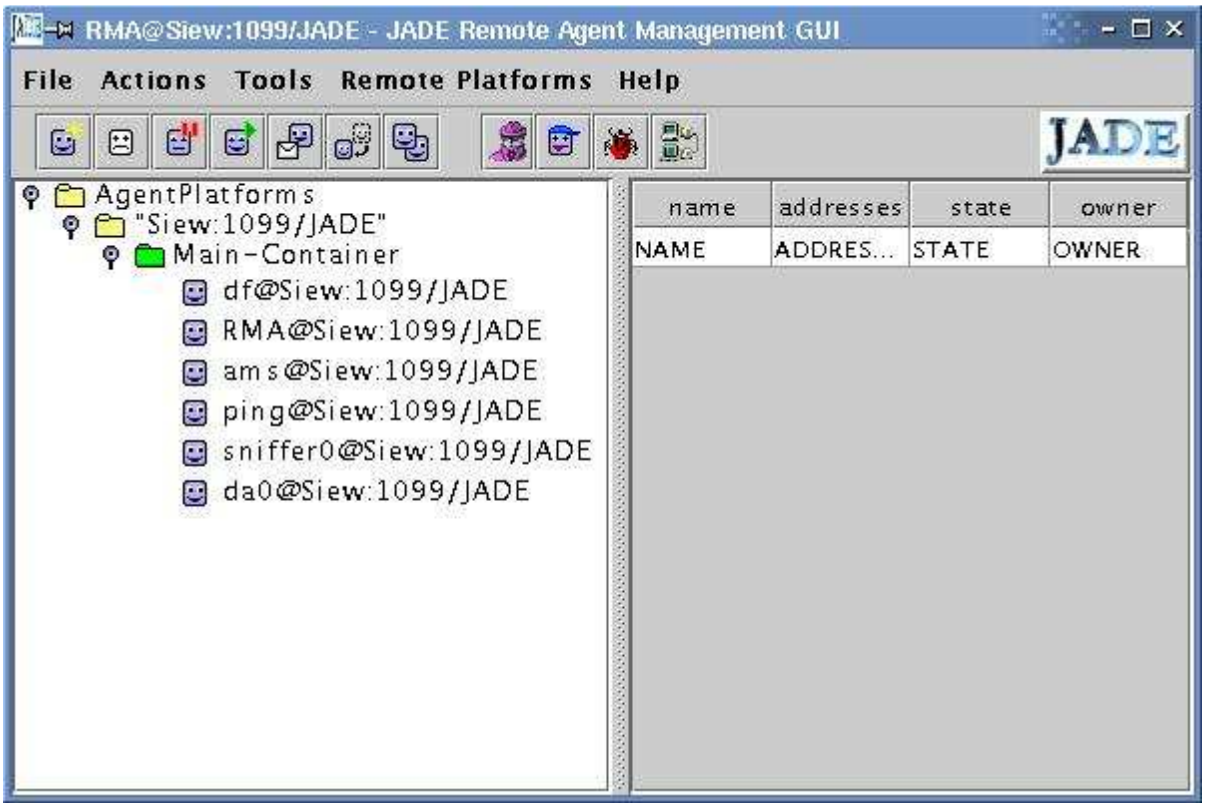
Tras hacer esto se debe iniciar la plataforma, creando al inicio un agente de tipo PingAgent. Para ello, ingresamos en el interior del directorio src y ejecutamos:

java jade.Boot -gui ping:examples.PingAgent.PingAgent

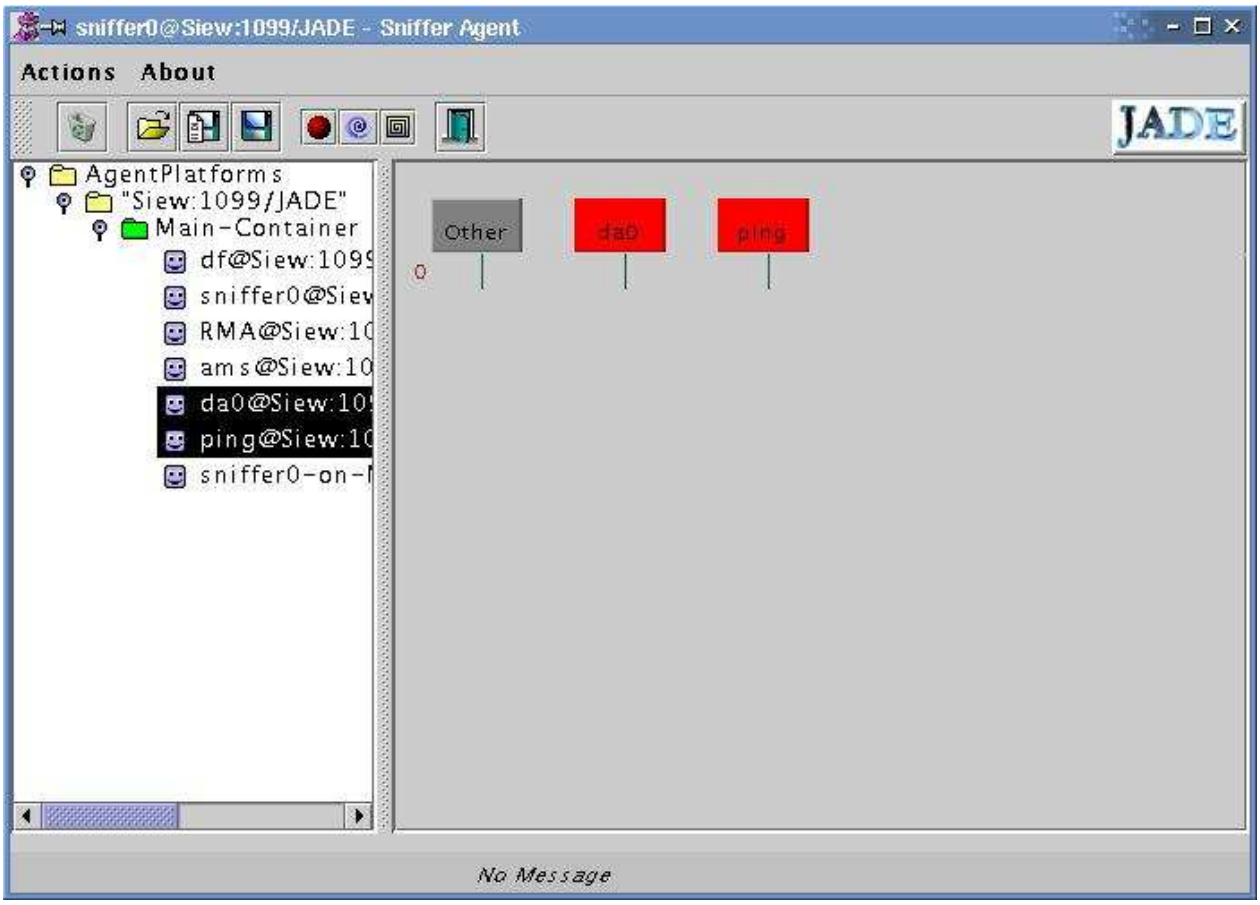
Como no se ha indicado nada con las opciones correspondientes, el agente ping será introducido en el contenedor Main-Container:



Para comprobar el funcionamiento de este agente, así como para practicar el uso de las herramientas SnifferAgent y DummyAgent, creamos un agente de cada uno de estos tipos, seleccionando las opciones correspondientes del menú Tools:

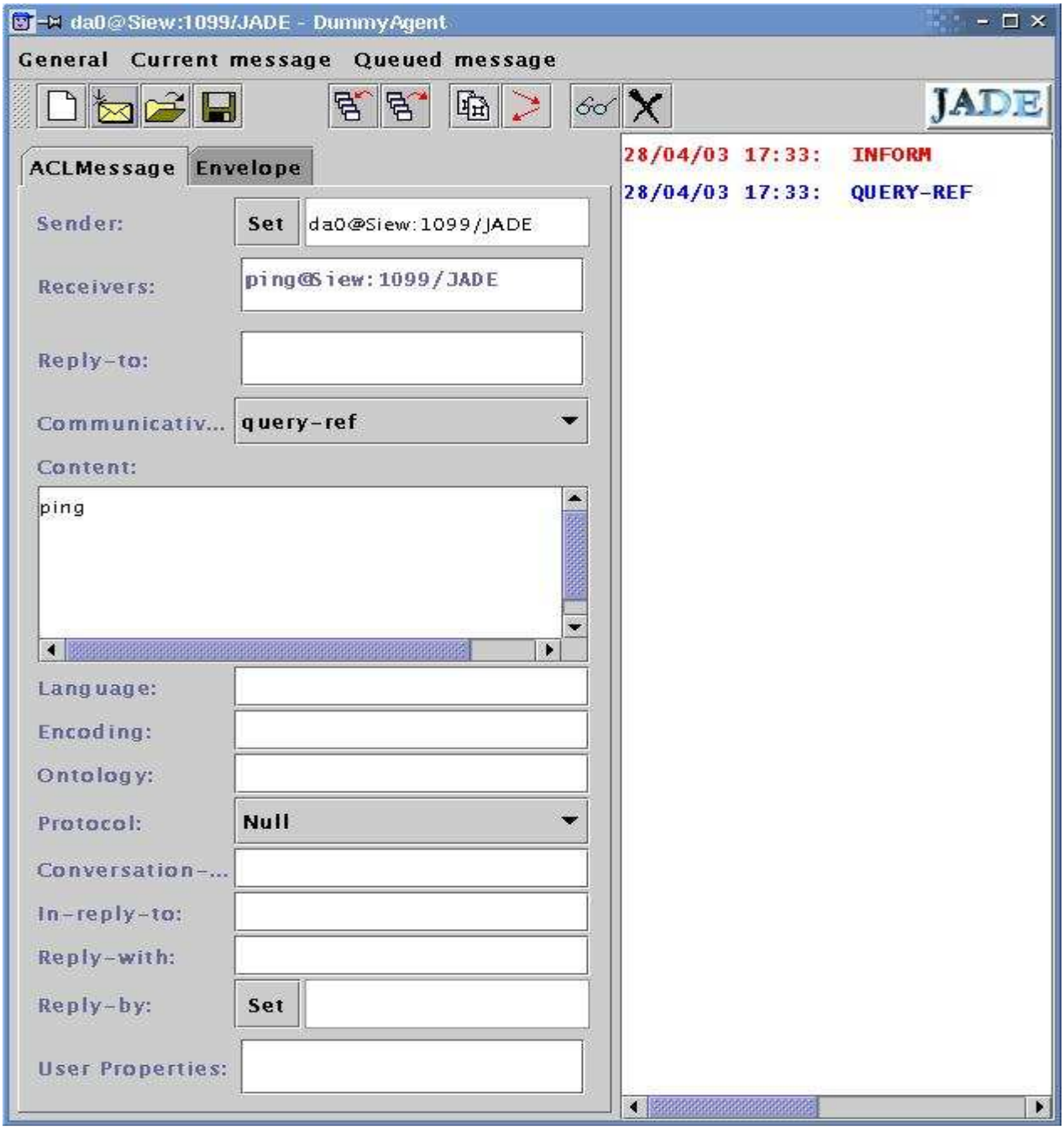


El agente Sniffer y el agente DummyAgent serán llamados sniffer0 y da0 respectivamente. Para visualizar gráficamente el intercambio de mensajes, en el diálogo del Sniffer seleccionamos los agentes ping y da0 (se pueden seleccionar varios agentes haciendo click sobre el nombre de cada uno mientras se mantiene pulsada la tecla CONTROL) y usamos la opción Do sniff this agent(s) del menú Actions. El aspecto del cuadro de diálogo será similar al siguiente:

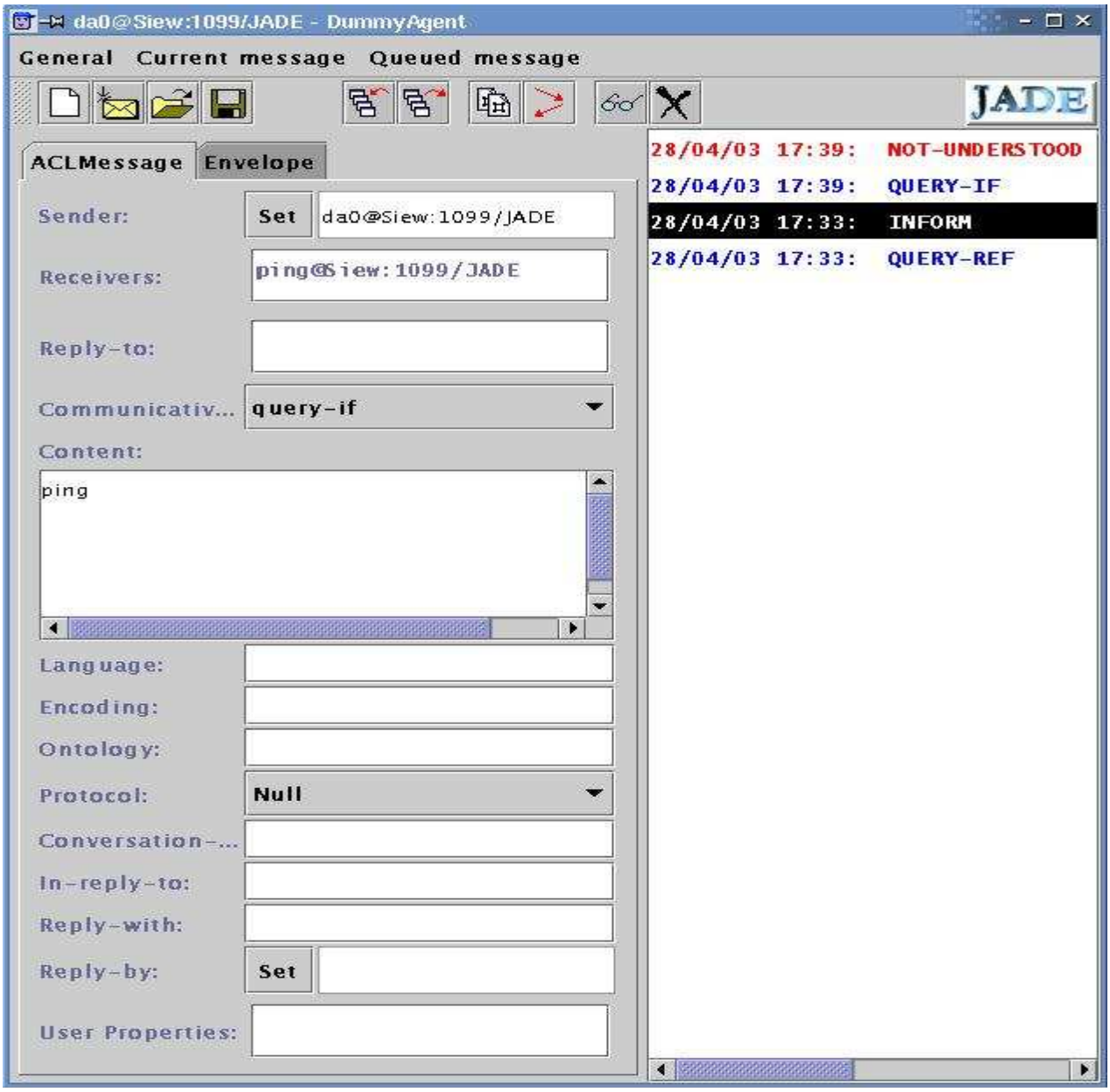


Ya está todo preparado para enviar el mensaje adecuado al agente ping por medio del DummyAgent da0. Dentro de la ventana asociada a dicho agente, pulsamos con el botón derecho sobre el campoReceivers e introducimos el nombre completo del agente ping (en el caso de este ejemplo, ejecutado sobre la máquina de nombre Siew, el nombre completo sería ping@Siew:1099/JADE). Este nombre debe ser introducido en el campo NAME de la ventana que se mostrará al pulsar con el botón derecho sobre dicho campo (el resto de campos se pueden dejar vacíos). En el campo Communicative act seleccionaremos la performativa QUERY-REF, y dentro del campo content introduciremos la palabra ping. Una vez rellenados los campos adecuados podemos enviar el mensaje con la opción correspondiente. Observaremos en la cola de mensajes de la parte derecha de la ventana el mensaje QUERY-REF enviado por el DummyAgent, así como la contestación del agente ping, un mensaje de

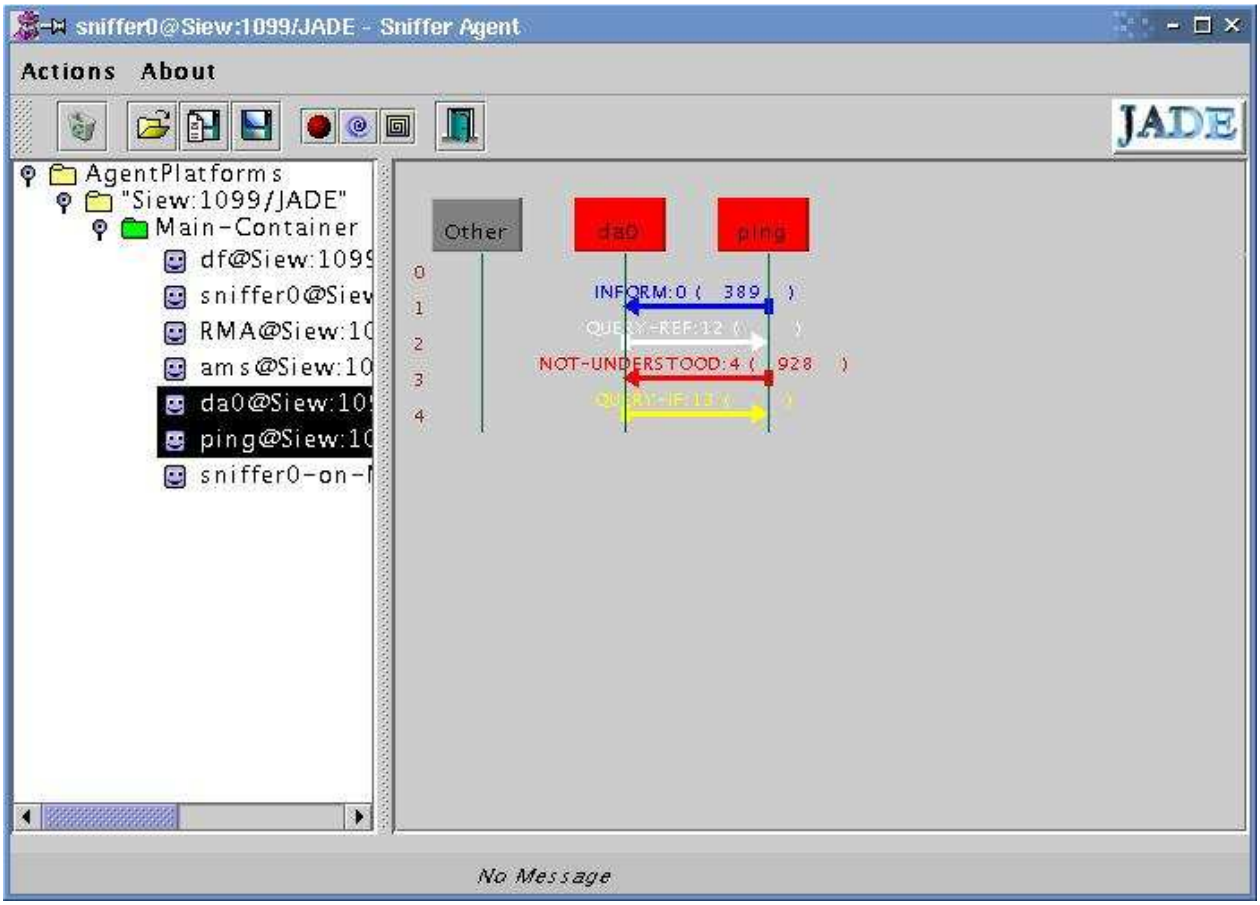
tipo INFORM (se muestran en distinto color, para diferenciar los mensajes enviados por el DummyAgent de las contestaciones recibidas).



Si seleccionamos el mensaje INFORM de la cola de mensajes y después la opción correspondiente a examinar el mensaje, podremos ver la contestación enviada por el agente *ping* (la palabra *alive*). Si ahora enviamos un mensaje con cualquier performativa que no sea QUERY-REF (por ejemplo QUERY-IF), veremos como el agente *ping* contesta con un mensaje NOT-UNDERSTOOD:



Mientras hemos hecho todo esto el agente *sniffer0*, ha ido representando gráficamente el intercambio de mensajes producido:



Haciendo doble-click sobre cualquiera de los mensajes (representados por flechas) podremos examinar el valor de cada uno de los campos del mensaje seleccionado.