UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

DEPARTAMENTO DE COMPUTACION Y SIMULACION DE SISTEMAS

"INTEROPERABILIDAD DE AGENTES MOVILES (IDEAS)"

ASESOR

PROF. VICTOR LOPEZ CABRERA Msc.

INTEGRANTES

JAVIER SANCHEZ GALAN **8-767-455**

JORGE MENDEZ **8-762-1802**

Trabajo de Graduación para optar a los Títulos de Licenciado en Ingeniería de Sistemas Computacionales y Licenciado en Redes Informáticas.

2006

Agradecimientos

El grupo de trabajo de esta investigación le agradece primeramente a Dios Todopoderoso por permitirnos vivir la experiencia de realizar un proyecto de investigación científica, darnos las fuerzas y suplir los recursos para concluirlo con éxito.

También agradecemos a las siguientes personas que de alguna forma u otra ayudaron a la terminación de esta investigación:

- *Dr. Ulrich Pinsfdorf y el equipo de trabajo de la plataforma SeMoA (Fraunhofer IGD, Alemania)*, por su explicación del funcionamiento del paquete de interoperabilidad entre las Plataformas SeMoA Y JADE.

- *Marcia Silvana Stippi (Universidad Nacional de la Patagonia San Juan Bosco, Sede Trelew, Argentina)*, por su ayuda incondicional al momento de aprender las reglas básicas para la utilización de la Plataforma JADE.

- *Jordi Cucurull Juan MSc. (Universidad Autónoma de Barcelona, España),* por su apoyo y explicación al momento de utilizar del Inter-Platform Mobility Service (IPMS) de JADE.

- *Dra. Oris Sanjur (STRI, Panamá),* por fungir como consejera y segunda opinión durante la realización de la investigación.

Dedicatoria

Javier:

A Dios Todopoderoso que cada día me da las fuerzas para seguir adelante y a quien le debo todo lo que tengo y soy.

A mi familia Rafael (Papa), Osmila (Mama) y Rafael (Hermano), que me han apoyado incansablemente con amor, a conseguir todo las metas que me he propuesto y que sin duda estarán ahí en las muchas que vienen.

"Mejor es adquirir sabiduría que oro preciado;
Y adquirir inteligencia vale más que la plata"
Proverbios 16:16

Jorge:

A Dios quien por medio de mis abuelos, padres, hermanos y amigos me ha enseñado el valor de la vida, la importancia de la superación personal, del trabajo bien hecho y de ayudar a mis semejantes.

Índice

Resumen Descriptivo

La interoperabilidad entre plataformas de agentes móviles representa uno de los tres factores más decisivos por lo cual la tecnología de agentes no ha tenido una gran acogida por los usuarios de las Tecnologías de Información y Comunicación (TIC).

Este trabajo muestra los avances que se han tenido a la fecha en el tema de interoperabilidad entre plataformas de agentes móviles de manera teórica y con pruebas experimentales.

Se realizaron pruebas de interoperabilidad utilizando las plataformas de agentes JADE y SeMoA. Además se realizaron pruebas de desempeño de migración de agentes entre plataformas JADE utilizando el módulo “Inter-Plataform Mobility Service”.

Introducción

En esta parte introducimos al lector a este trabajo de graduación explicando el contexto de la investigación, luego se explica nuestras motivaciones y metas y por último una descripción de los capítulos de este trabajo.

Contexto de la Investigación

Este trabajo de investigación se llevó a cabo en la Unidad de Sistemas Inteligentes (USI) de la Facultad de Ingeniería de Sistemas Computacionales de la Universidad Tecnológica de Panamá, unidad de investigación que está bajo la coordinación del profesor Víctor López, quien también fungió como asesor de este trabajo.

La USI tiene entre sus objetivos investigar todo tipo de sistemas inteligentes, haciendo énfasis en la utilización de agentes inteligentes de software, los cuales representan el fin último de la inteligencia artificial. En la unidad se han llevado a cabo investigaciones en las áreas de sistemas multiagentes con agentes colaborativos, agentes móviles (software), sistemas de seguridad biométricos y robótica.

Motivación y Metas

La tecnología de agentes es un paradigma novedoso que propone nuevas formas de desarrollo de sistemas y aprovecha al máximo los sistemas distribuidos, redes y capacidad de procesamiento actual de nuestros equipos informáticos.

A pesar de tantos beneficios, en esta tecnología se han detectado tres problemas básicos que esta tecnología debe superar: *la administración*, *la seguridad* y *la interoperabilidad* entre Plataformas de Agentes.

Este trabajo tiene como meta mostrar *el estado del arte de la interoperabilidad entre los sistemas de agentes móviles* y deja para trabajos posteriores los temas de administración y seguridad por su nivel de complejidad.

Las siguientes interrogantes conforman nuestras preguntas de investigación:

- ¿Qué es un agente inteligente de software y qué es un agente móvil?
- ¿Qué es una Plataformas de Agentes, Cuales son las Plataformas Existentes en la actualidad y cuales son sus limitaciones y ventajas?
- ¿Qué es la Interoperabilidad entre Plataformas de Agentes?
- ¿Existen reglas o estándares para crear Plataformas de Agentes?
- ¿Es posible hacer que dos Plataformas de Agentes sean Interoperables y si es posible, ¿Cuáles son los métodos para hacerlo?

Con este trabajo logramos responder estas interrogantes en base a documentación teórica y también hemos logrado comprobar formulaciones experimentalmente al replicar pruebas de interoperabilidad entre dos plataformas de agentes SeMoA y JADE. Además de esto llevamos a cabo con éxito pruebas de desempeño con agentes móviles en la plataforma JADE.

Descripción de los Capítulos

La estructura de este documento es la siguiente:

- *Capítulo I:* una introducción a la tecnología de agentes explicando su historia, sus bases filosóficas, los avances existentes y lo problemas que enfrenta en la actualidad.

- *Capítulo II:* una explicación detallada de los problemas existentes en la tecnología de agentes, los estándares que la rigen e indica las características de las plataformas de agentes existentes y especifica las de las dos escogidas para esta investigación.

- *Capítulo III:* provee los fundamentos de la interoperabilidad entre plataformas de agentes, haciendo énfasis en los puntos de ejecución, comunicación y migración de agentes en las plataformas SeMoA y JADE.

- *Capítulo IV:* presenta las pruebas que realizamos en esta investigación, detalla la metodología seguida y resultados de las pruebas interoperabilidad y pruebas desempeño y termina con las conclusiones obtenidas luego del análisis de los resultados de las pruebas para luego finalizar con las recomendaciones para trabajos futuros.

1.1 Computación en el Siglo 21

La computación como tal, ha tenido muchos significados, en los años 40's y 50's, computación era sinónimo de *cálculo,* generalmente cálculos numéricos, las computadoras eran utilizadas para resolver grandes problemas matemáticos o físicos.

En los 70's y 80's con los avances en los medios de almacenamiento de datos y manipulación digital de información, computación era un sinónimo de *Procesamiento de información,* generalmente se utilizó para transacciones bancarias y todo tipo de sistemas de información gerencial.

En los últimos 15 años, con el crecimiento del Internet y del World Wide Web en general, hemos llegado al punto en que la computación ha cambiado nuevamente de significado, esta vez podríamos decir que computación es sinónimo de *interacción*, ya que la computación es algo que ocurre gracias a la comunicación entre dos entidades computacionales. [2]

En el pasado la computación se distinguía por ser una actividad solitaria (involucraba un sistema y un pequeño grupo de computadoras), en nuestros días la computación es una actividad social (involucra muchos sistemas y muchos grupos de computadoras), por esto han aparecido nuevas formas de concebir, diseñar, desarrollar y manejar los sistemas computacionales.

Un ejemplo claro de las nuevas formas de concebir los sistemas ha sido la arquitectura orientada a servicios, este tipo de arquitecturas podemos definirlas como:

"Un modelo de componentes que interrelaciona las diferentes servicios de una aplicación mediante interfaces bien definidas. Donde los servicios están débilmente acoplados, pero son altamente interoperables". [6]

Las arquitecturas orientadas a servicios están conformadas por componentes de muchos sistemas que son capaces de interoperar entre sí proveyéndose servicios; al contrario de los sistemas monolíticos en donde existe un único sistema cerrado que realiza todas las funciones, con un nivel mínimo o inexistente de interoperabilidad.

Estos componentes de las arquitecturas orientadas a servicios tienen ciertas características como:

- Lo más probable es que los componentes no hayan sido diseñados por el mismo grupo de desarrollo de software.

- Los componentes y sus servicios pueden ser administrados por una organización, pero formar parte de otra, haciendo necesario el acceso a varias fuentes de información, como ejemplo los *sistemas en Grid*, que son sistemas que tienen la habilidad de procesar grandes volúmenes de información, distribuyéndolos en procesos a los computadores de una red, creando una arquitectura virtual.

- Los componentes no son necesariamente activados por usuarios humanos, sino que ciertas acciones son coordinadas automáticamente, bajo ciertas condiciones, como ejemplo los *sistemas Ad hoc de redes inalámbricas*, que son sistemas de redes inalámbricas auto configurables de topología arbitraria.

- Componentes autónomos o inteligentes pueden llevar a cabo el ensamblaje de software o de sistemas, para permitir respuestas a circunstancias internas o externas, estos sistemas se asemejan a los sistemas naturales o a las sociedades humanas y por lo tanto las ideas de biología, física estadística, sociología y economía han tomado valor en el desarrollo de sistemas computacionales.

La pregunta que salta de nuestras mentes inmediatamente es:

¿Cómo sacarle provecho a la computación como interacción, sabiendo que esta implica actividad social entre entidades inteligentes independientes, que se adaptan y cooperan entre sí?

Muchas personas creen que la respuesta a esta pregunta es la **Tecnología de Agentes**.

1.2 ¿Qué es la Tecnología de Agentes?

Los sistemas basados en agentes han sido una de las más importantes áreas de investigación y desarrollo de la tecnología de información desde el principio de los años 90's. Algunos estudiosos creen que la tecnología de agentes es uno de los paradigmas más importante en computación después de la aparición del paradigma de programación orientada a objetos. [2]

Desde un punto de vista de evolución tecnológica, se puede decir que los conceptos que constituyen la teoría de agentes derivan del desarrollo de áreas de la informática muy diferentes, entre ellas: redes de computadoras, ingeniería de software, inteligencia artificial, interacción humano-computadora, sistemas distribuidos y concurrentes, sistemas móviles, telemática, trabajo cooperativo asistido por computadora, sistemas de control, sistemas de toma de decisiones, búsqueda y manejo de información y comercio electrónico.

Los orígenes de la tecnología de agentes comienzan con la Inteligencia Artificial Distribuida (IAD), pero podemos decir que el concepto de agente como entidad computacional aislada se tiene, evolucionando desde la IAD, debido a conceptos de la Ingeniería del Software.

El desarrollo práctico de los agentes se ha visto potenciado por el enorme crecimiento de Internet. Dentro de este ámbito aplicado existe una gran influencia de la teoría de interfaces de usuario y de la interacción hombre-máquina, entre los que cabe destacar aquellos aspectos relacionados con el acceso inteligente a las fuentes de información en Web y la necesidad de programas en los que el usuario sea capaz de delegar el trabajo y la toma de decisiones (por ejemplo, en actividades de comercio electrónico).

En la Figura 1.1 se representan esquemáticamente las relaciones entre las principales áreas que conforman la base de la teoría de agentes.

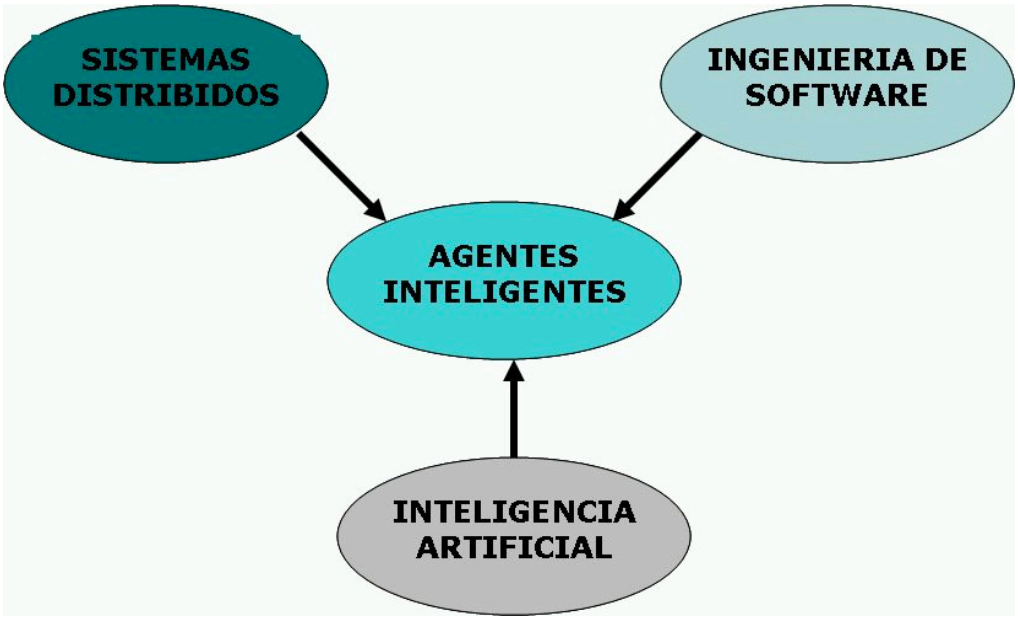


Fig. 1.1 - Áreas relacionadas con los Agentes Inteligentes de Software. Fuente: Los Autores.

La base de la Tecnología de Agentes, *son los agentes*, que en el sentido más simple, son programas capaces de tener acciones autónomas en dominios dinámicos impredecibles, típicamente sistemas multiagentes.

1.3 ¿Qué es un Agente Inteligente?

Antes de entrar a la definición computacional del término agente, es pertinente saber la definición según el diccionario La Real Academia de la Lengua Española, el cual lo define como:

Agente.

(Del latin agens, -entis, voz activa de agere, ***hacer****).*

1. adj. Que obra o tiene virtud de obrar.

2. m. Persona o cosa que produce un efecto.

3. m. Persona que obra con poder de otra.

4. com. Persona que tiene a su cargo una agencia para gestionar asuntos ajenos o prestar determinados servicios.

Partiendo del significado podemos decir que el agente inteligente es una abstracción del agente del mundo real, ya que como vemos en el significado, un agente es capaz de obrar en nombre de otra persona, producir efectos sobre el ambiente donde esta y es capaz de prestar servicios a las entidades que conforman su ambiente.

1.3.1 Características de los Agentes Inteligentes

Entre los primeros autores en dar una definición del agente inteligente y sus características y funciones que lo definen como tal, podemos destacar a Wooldridge y Jennings [46] quienes distinguen dos usos generales del término agente:

- *Noción débil de agencia*: sistema de hardware o un sistema de cómputo basado en software que contiene las propiedades de autonomía, habilidad social, reactividad y proactividad. Características que podemos definir como:

- *Autonomía*: los agentes actúan sin la intervención directa de las personas. Posee la capacidad de razonamiento para generar cursos de acción.

- *Sociabilidad*: los agentes interactúan con otros agentes mediante algún mecanismo de comunicación.

- *Reactividad*: los agentes perciben su ambiente (mundo real, interfaz gráfica, conjunto de otros agentes, Internet o combinación de estos) y responden a los cambios de éste.

- *Proactividad*: los agentes no solo actúan en respuesta a su ambiente, sino que son capaces de tener comportamiento orientado a metas.

- *Noción fuerte de agencia*: sistema computacional que, además de las propiedades mencionadas en la noción débil de agencia, es conceptualizado e implementado usando conceptos que son más usualmente aplicados a humanos, como lo son: creencias, intenciones, obligación, capacidad de cooperación y aprendizaje, movilidad, veracidad, benevolencia y racionalidad. Características que podemos definir como:

  - *Cooperación:* en consecuencia a la sociabilidad, la cooperación entre agentes es una razón de ser para tener múltiples agentes para resolver problemas.

  - *Aprendizaje:* los agentes para ser inteligentes requieren la propiedad de poder aprender del ambiente que les rodea.

- *Veracidad:* un agente no comunica deliberadamente información falsa.

- *Benevolencia:* los agentes no tienen metas conflictivas, hará siempre lo que se pida.

- *Racionalidad:* un agente actuará para alcanzar sus metas en la medida que sus creencias, conocimientos y capacidad de razonamiento se lo permitan.

- *Movilidad:* algunos agentes tienen la habilidad de viajar en una red de computadoras (en Internet).

1.3.2 Clasificación de los Agentes Inteligentes

Los agentes pueden clasificarse desde diferentes puntos de vista que se mencionan a continuación:

- En base a sus capacidades de resolver problemas se clasifican en:

  - *Agentes reactivos:* reaccionan a cambios en su ambiente o a mensajes provenientes de otros agentes. No son capaces de razonar acerca de sus intenciones. Sus acciones se realizan como resultado de reglas que se establecen anteriormente.

  - *Agentes intencionales*: son capaces de razonar acerca de sus intenciones y conocimientos, crear planes de acción y ejecutar dichos planes. Los agentes intencionales pueden ser considerados como sistemas de planeación.

  - *Agentes sociales:* poseen la capacidad de los agentes intencionales. Mantener los modelos de los otros agentes,

razonar sobre el conocimiento incorporado a estos modelos. de otros agentes.

- En base a autonomía, aprendizaje y cooperación:

  - *Agentes colaborativos*: enfatizan su autonomía y cooperación (con otros agentes) para realizar su tarea. Pueden aprender.

  - *Agentes de interfaz*: ponen énfasis en su autonomía y aprendizaje para realizar sus tareas. Los agentes de interfaz asisten y dan soporte al usuario para aprender el uso de una aplicación.

- En base a su movilidad:

  - *Agentes estáticos*: son programas que ejecutan tareas simples establecidas por el usuario, el ejemplo más conocido son los asistentes personales (asistentes electrónicos como el clip de Microsoft Office). También existen otros que son programados para tareas como búsqueda de información y compras en línea. Los agentes estáticos pueden ser y no ser sociales y pueden o no aprender.

  - *Agentes móviles*: los agentes móviles son programas de software capaces de viajar por redes de computadoras (como Internet), son capaces de interactuar con computadoras remotas, pedir información a nombre de un usuario y regresar a su lugar de origen una vez que ha realizado las tareas especificadas por un usuario.

  - *Agentes híbridos:* son aquellos que en su funcionamiento poseen una combinación de capacidades, están programados para

reaccionar como estáticos o como móviles de acuerdo con la situación.

1.3.3 Otras Clasificaciones de Agentes Inteligentes

En la tabla siguiente se describen muchas de las características por medio de las cuales se clasifican los agentes.

Característica	Significado
Reacción	Responde a cambios en el ambiente
Autonomía	Tiene el control de las acciones que realiza
Orientado a tareas (proactivo)	No reacciona a los cambios del ambiente, si el actuar no le influye a la consecución de sus tareas u objetivos
Continuo en el tiempo	Es un proceso en ejecución continua
Comunicación (sociable)	Capaz de comunicarse con otros agentes y hasta con seres humanos
Aprendizaje (adaptativo)	Cambia su comportamiento basado en experiencias previas
Movilidad	Capaz de moverse de un lugar a otro (de una computadora a otra)
Flexibilidad	Sus acciones no están escritas en piedra
Carácter	Tienen personalidad y estados de ánimo

Tabla 1.1 – Características del Agente Inteligente. Fuente: Stan, F.; Art, G. Is it an Agent, or just a Program?.

A los agentes se les pueden clasificar de acuerdo a un subconjunto de propiedades que tenga, por definición el agente tiene al menos cuatro

propiedades (reacción, proactividad, autonomía y continuidad en el tiempo), al agregar otras características producen otras subclases de agentes. [43]
Son posibles otras clasificaciones, como clasificar los agentes de acuerdo con:

- Las tareas que realizan, como ejemplo los agente de planeación
- El rango y la sensitividad de sus sensores
- Rango y efectividad de sus acciones

La taxonomía de los agentes de software según Brustoloni [11], los divide en tres grandes grupos: los agentes de regulación, los agentes de planeación y los agentes adaptativos.

- *Un agente de regulación:* también llamado agente reactivo, reacciona a cada entrada en su sistema y siempre sabe que hacer. No planea ni aprende. Un ejemplo de este tipo de agente es la regulación de la temperatura hecha por un termostato o algún método de regulación homeostático del cuerpo de un animal o del cuerpo humano. (Podemos definir homeostasis como el proceso de adaptación con el ambiente)
- *Un agente de planeación:* planea que hacer usando Inteligencia Artificial por medio de casos, métodos de búsqueda y métodos de mezcla.
- *Un agente adaptativo:* no solo planea, sino que aprende.

Una última clasificación puede involucrar al ambiente (contexto) donde se encuentra el agente.

1.3.4 Los Agentes Inteligentes de Software

Los agentes de software se distinguen de los objetos (objetos del paradigma de orientación a objetos) en que los primeros son entidades autónomas capas de decidir sobres sus acciones e interacciones y proceden a lograr sus objetivos individuales.

Los agentes son capaces de ejercer su autonomía escogiendo cómo realizar las tareas asignadas a ellos, aun más importante ellos toman decisiones basadas en el ambiente donde estén.

De tal manera que los agentes no pueden ser invocados como objetos, pero se les puede asignar tareas a realizar. A pesar de esto no debemos confundir que los agentes pueden ser construidos basados en muchas tecnologías, incluyendo la orientación a objetos o como servicios Web.

Es posible encontrar varias definiciones sobre qué es un agente inteligente de software, entre ellas:

"Los agentes de software pueden definirse como entidades computacionales autónomas que resuelven problemas, capaces de operar efectivamente en ambientes abiertos y dinámicos. Los agentes son puestos con frecuencia en ambientes donde deben interactuar y cooperar con otros agentes (y con personas), este tipo de sistemas son llamados sistemas multi-agentes." [3]

Para los fines de esta investigación hemos acuñado la siguiente definición, para el concepto de agentes inteligentes computacionales:

"Un agente inteligente en términos de computación es un programa que representa a una entidad del mundo real (a una persona o entidad computacional), en el mundo virtual (redes de computadoras), que además posee un comportamiento autónomo, es capaz de decidir, reaccionar,

comunicarse con otros agentes y aprender del ambiente (adaptarse) en donde es ejecutado."

1.3.4.1 Agentes Estáticos

La movilidad *NO* es una cualidad necesaria para que los *programas* sean llamados agentes, existen muchas aplicaciones en las que no es necesario que el agente se mueva luego de su ejecución inicial, a estos agentes se les llama agentes estáticos y gracias a propiedades como ejecución asincrónica y autónoma (independiente de los demás procesos del sistema), llevan a cabo tareas complicadas que no requieren movilidad.

Como un ejemplo práctico del funcionamiento de los agentes estáticos podemos analizar los Applets de JAVA, los cuales luego de ser enviados a una maquina destino para su ejecución en la maquina virtual de JAVA (JVM)[1], no se mueven a otras maquinas luego de terminar su ejecución.

Si un agente estático necesita información de otro sistema o se necesita comunicar con otro agente en un sistema remoto, debe buscar algún método de comunicación remota como "Llamadas a Procedimientos Remotos" (RPC)[2].

1.3.4.2 De Agentes Inteligentes a Agentes Móviles

Poco a poco en la tecnología de agentes se ha ido transformando la importancia del atributo inteligencia por el de movilidad. Por mucho tiempo se ha analizado y pensado la manera de cómo crear agentes muy inteligentes, agentes capaces de pensar utilizando inferencias como lógica de primer orden (también llamada lógica de predicados; donde un predicado es una función con argumentos que se puede evaluar a verdadero o falso) y lógica multimodal

[1] La JVM es una máquina virtual que ejecuta el código resultante de la compilación de un programa escrito mediante el lenguaje de programación Java.
[2] RPC es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

o BDI (Los agentes se suelen considerar como sistemas cuya conducta se puede predecir atribuyendo Creencias, Deseos e Intenciones o conducta racional), lo cierto que estos tipos de desarrollos son muy costosos (en horas-hombre y en tiempo computacional) y en muchas ocasiones con pobres resultados.

Al contrario de los agentes muy inteligentes (o agentes pensantes), crear agentes móviles es relativamente sencillo y para ciertas aplicaciones estos brindan grandes beneficios y mejoran el desempeño global de una aplicación (generalmente en sistemas distribuidos).

La movilidad cada día juega un papel más relevante en el desarrollo de sistemas, por lo que muchas aplicaciones de agentes estáticos existentes han creado nuevas versiones incluyendo agentes móviles y otros sistemas se han desarrollado de nuevo para incluir movilidad.

Mientras los agentes estáticos son útiles en cierto tipo de aplicaciones, es evidente que proporcionar movilidad a los agentes incrementa la flexibilidad del sistema.

Un agente móvil es capaz de decidir si mover la información hacia sí o moverse hasta donde esta la información. [10]

En los últimos años el crecimiento del Internet ha traído consigo problemas como: el tráfico excesivo y la realización de tareas rutinarias, para resolver estos y otros problemas los agentes móviles se han presentado como una gran alternativa de solución.

1.4 Agentes Móviles

Como fue mencionado previamente, los agentes inteligentes se clasifican según su movilidad en agentes estáticos y móviles. A continuación una definición ampliada sobre que son los agentes móviles:

*"Los Agentes móviles son **programas nómadas** que actúan como representante de una persona en el mundo virtual, capaces de moverse autónomamente a través de redes. Son capaces de visitar nodos de red, usando la capacidad de computación del nodo e independientes del sistema operativo."* [10]

1.4.1 Descripción

Los agentes móviles pueden ser descritos como: código de programación, datos y estados de ejecución, que migran de un servidor de agentes (agencia) hacia otro. Accediendo los recursos localmente, ellos son capaces de colectar, procesar y publicar información dentro de una red. Los agentes móviles llevan los sistemas distribuidos al límite, ya que no solo los procesos son distribuidos dinámicamente, sino que también el código que los ejecuta también es distribuido. Los agentes móviles ofrecen un gran beneficio a las aplicaciones de red, ya que añaden inteligencia del lado del cliente y funcionalidad a los servicios del lado del servidor [36].

La característica principal del agente móvil es tener métodos invocables que le permite moverse de un nodo a otro. El agente posee un programa de ejecución principal que define las tareas que realizará y el grado de inteligencia para interactuar con los agentes (y los seres humanos) y para resolver las tareas (o inconvenientes) que le presente su entorno de ejecución. [35]

En el punto 1.3 (¿Qué es un Agente Inteligente?) de este capítulo se expuso todas las características de un agente inteligente y sus clasificaciones; pero es necesario conocer las características básicas de una agente móvil, entre ellas:

- *Autonomía*: El agente móvil es autónomo, ya que no necesita supervisión humana para tomar una decisión y porque posee un hilo de ejecución propio (thread) e independiente del resto de los agentes.

- *Móvil:* serializable. Este concepto se explicara a fondo más adelante en este capítulo.

- *Concurrente:* puede ejecutarse con más de un agente a la vez (generalmente en sistemas multiagente).

- *Direccionable*: se sabe definir su comportamiento.

- Continuo: se ejecutan continuamente y por tiempo indefinido

- Reactivo: reacciona a su entorno mediante métodos

- Social: interopera con otros agentes y con servicios.

- *Adaptativo:* controla sus acciones mediante excepciones (excepciones o instrucciones del tipo try-catch)

1.4.2 Fundamentos de los Agentes Móviles

Debido a la arquitectura acefálica de Internet (carente de centro o de principio y fin) y del crecimiento exponencial del numero de usuarios por cada año que pasa, era necesario un nuevo paradigma que satisficiera dos necesidades

básicas: incrementar los tipos posibles de comunicación y capaz de no saturar las redes disminuyendo el consumo de ancho de banda.

1.4.2.1 Arquitectura Cliente-Servidor

El principio central de las comunicaciones en las redes de computadoras, son las "Llamadas a procedimientos remotos" RPC, los cuales permiten a una computadora llamar procedimientos en otra. En la figura 1.2 se aprecia un diagrama de un cliente haciendo llamadas a procedimientos remotos en un servidor.

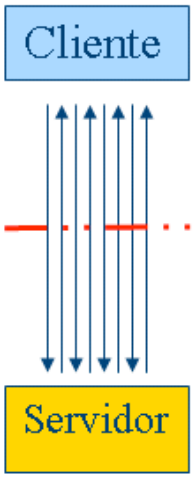


Fig. 1.2 – Arquitectura cliente servidor y paso de mensajes a través de conexiones de red. Fuente: Los Autores.

Cada mensaje en la red transporta solicitudes o confirmación de procedimientos (en ambas direcciones). Una solicitud incluye datos que son los argumentos del procedimiento y la respuesta incluye datos que son resultados. El procedimientos mismo es interno a la computadora que lo ejecuta y los mensajes viajan desde el punto origen al punto destino.

1.4.2.2 Arquitectura de Agentes Móviles utilizando el paradigma de Programación Remota

Una alternativa a las llamadas a procedimientos remotos (RPC) es la "***programación remota***"; en la que dos servidores que se comunican con el paradigma de programación remota hacen un acuerdo sobre las instrucciones que son permitidas en un procedimiento y los tipos de datos permitidos en su estado. Estos acuerdos constituyen un lenguaje (ontología). El lenguaje incluye las instrucciones que permiten al procedimiento tomar decisiones, examinar y modificar su estado y llamar procedimientos proporcionados por la computadora que está recibiendo el programa.

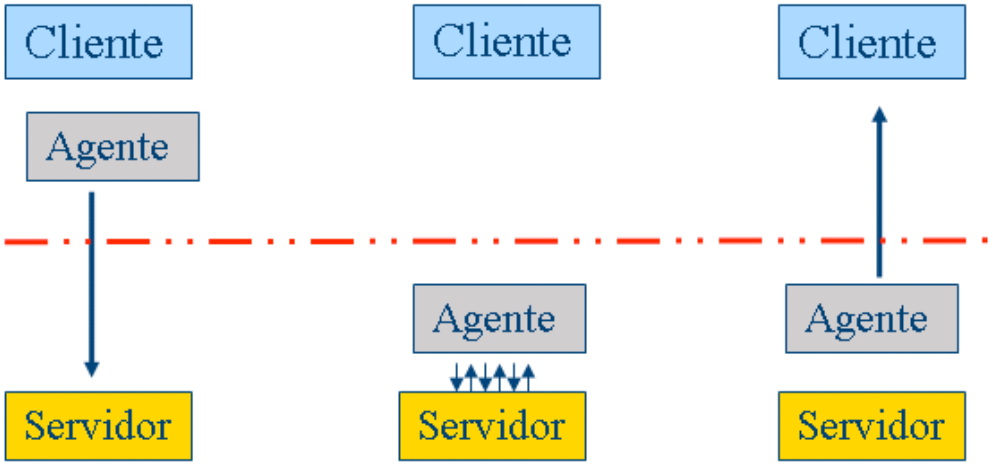


Fig. 1.3 – Arquitectura de programación remota mediante agentes. Sólo existen dos conexiones de red, al enviar el agente y al recibirlo de vuelta. Fuente: Los Autores.

Tales llamadas a procedimientos son locales en vez de remotos. En la figura 1.3 se aprecia como el proceso de llamada de procedimientos es local entre el agente y el servidor. El procedimiento y su estado son llamados "***agente móvil***", ya que éste representa a la computadora que los envía, aunque ellos residen y operan en la computadora que los recibe.

1.4.2.3 Capacidad de Movilidad (Serialización)

Mientras que el paradigma de llamadas a procedimientos remotos (RPC) sólo permite *movilidad de código*, el concepto de agentes móviles permite *movilidad de procesos*. [26]

Un agente móvil no solo carga consigo su código de ejecución sino también información de estado del agente. El estado del agente consiste en dos partes: sección de datos y el estado de ejecución.

- Dentro de la *sección de datos* se encuentran las variables globales y variables instanciadas.

- Dentro del *estado de ejecución* se encuentran las variables locales y los hilos de ejecución activos.

La capacidad de movilidad de una agente puede ser clasificada en dos nociones, noción débil y noción fuerte.

La noción fuerte de movilidad se da cuando el estado completo del agente es capturado (el hilo de ejecución)**,** *serializado* y transferido con el resto del código hacia el siguiente destino. Tan pronto cuando llega a su próximo destino el hilo del agente es ejecutado.

La noción débil de movilidad se da cuando se transfieren en forma *serializada* hacia el siguiente destino solamente el estado del agente (estado interno y datos privados) y/o el código del agente. Tan pronto como llega a su próximo destino el agente es reconstruido.

En las ciencias computacionales la serialización se define como el proceso por medio del cual se guarda el estado interno de un objeto, para luego transmitirlo a través de una conexión de red, en forma de una serie de bytes (o

en algún formato entendible por los seres humanos como XML). Cuando es recibido esta serie de bytes se utiliza para recrear el objeto (se crea un clon con idéntico estado interno que el objeto original).

Es necesario explicar que el concepto de serialización no es único del paradigma de agentes, sino que es un paradigma computacional muy conocido y aplicado. Como ejemplo cada vez que mandamos un archivo de *X* cantidad de bytes por medio de una red de computadoras, es necesario transferir *N* paquetes de *Y* cantidad de bytes, que luego de su transferencia sean capaces de ser ordenados y vueltos al estado original en el cual estaba en el lugar desde donde fue enviado.

En la tabla 1.2 se muestran ascendentemente los distintos grados de movilidad divididos por paradigmas (en el primer caso se mueven o transportan menos bytes y se va elevando la cantidad).

Paradigma Computacional	¿Qué es lo que realmente ocurre?
Paso de Mensajes	Transporte de Datos
RPC	Transporte de Código y Datos
Programación Remota con Agentes Móviles (Noción débil de Movilidad)	Migración de Código y Datos del agente
Programación Remota con Agentes Móviles (Noción fuerte de Movilidad)	Migración de Código, Datos y Estados del agente

Tabla 1.2 – Grados de Movilidad. Fuente: Schrool, Andre Peter. An Agent Architecture for Mobile Network Services: Design and Implementation

1.4.3 Beneficios de los Agentes Móviles

La ventaja directamente visible al utilizar programación remota con agentes móviles, es el mejoramiento en el tiempo de desempeño, debido a que es más eficiente enviar un agente a un servidor y trabajar localmente, en vez de trabajar remotamente.

Otra ventaja estratégica de la programación remota con agentes móviles es la personalización. Los componentes del servidor de una aplicación basada en programación remota, son dinámicamente instalados por la misma aplicación que se está ejecutando desde el ordenador del usuario, dado que cada componente es un agente.

Existen aplicaciones en las cuales la utilización de agentes móviles no es una ventaja visible, mientras que existen muchas otras que se pueden beneficiar grandemente utilizándolos.

Según Lange existen siete beneficios principales como resultado de la utilización de agentes móviles [25]:

1. *Reducen la carga de la red:* un agente se puede mover a una computadora destino y realizar las actividades localmente, en vez de transmitir los datos a través de la red.

2. *Reducen el tiempo de espera (latencia):* mantener los controles en sistemas grandes se puede transformar en una problemática debido a la latencia. Para resolver este problema un agente se puede mandar para hacer la tarea de revisión local como si se estuviera trabajando en un sistema en tiempo real (en vivo).

3. *Encapsulan protocolos*: los agentes se comunican con servidores u otros agentes usando sus propios protocolos y no necesitan utilizar los

métodos nativos de cada computadora (como comunicación entre procesos (IPC).

4. *Su ejecución es asincrónica y autónoma*: una vez que una agente migra de una computadora a otra se convierte en una entidad independiente.

5. *Se adaptan dinámicamente*: los agentes pueden percibir el ambiente y actuar por si mismos para resolver un problema.

6. *Son por naturaleza heterogéneos*: los agentes son dependientes de su ambiente de ejecución y no del software o hardware en el cual se ejecutan.

7. *Son robustos y tolerantes a fallos*: es posible utilizar la habilidad de migración de los agentes para implementarlos en sistemas en los que sea necesario la tolerancia a fallos.

1.4.4 Áreas de Aplicación de los Agentes Móviles

Los agentes móviles son útiles en muchas áreas, entre ellas:

- Aplicaciones de Usuario
  - Agentes de filtrado y búsqueda de información
  - Asistentes personales
- Sistemas Middleware (Sistemas Intermediarios)
  - Sistemas de archivo globales
  - Sistemas de colaboración distribuidos y de flujo de trabajo (workflow)
- Administración de Sistemas
  - Monitoreando el estado y control de la red
  - Detección de intrusos
  - Distribución de software, instalación y actualizaciones.

  - Control remoto de Sistemas
  - Sistemas Dinámicos
  - Servidores universales
- Correo activo
  - Enviar contenido ejecutable como correos

1.5 Sistemas Multiagente

Cuando hablamos de un sistema basado en agentes nos referimos a un sistema multiagente (SMA), con varios tipos de agentes (estáticos, dinámicos, reactivos, deliberativos, etc.) y con un número también variable de agentes de cada tipo.

Algunos agentes pueden ser bastante sencillos, de naturaleza reactiva. Sin embargo es normal que cuando el sistema proporciona cierto comportamiento inteligente (por ejemplo, capacidad de planificar, toma de decisiones, delegación de tareas, descomposición de objetivos, búsqueda de alternativas) exista algún agente con un potente modelo cognitivo.

Para programar esos comportamientos es necesario utilizar algo más que los lenguajes procedimentales clásicos. De modo que en la arquitectura de algunos sistemas multiagentes se han integrado mecanismos de razonamiento basados en reglas o inferencias.

De forma esquemática, los sistemas multiagentes que consideramos constan de:

- *Agentes:* de diversos tipos, que cooperan para proporcionar a los usuarios servicios de gran valor añadido, adaptables y personalizables.

Cada agente es un sistema encapsulado y con límites bien definidos, capaz de una acción autónoma y flexible en su entorno para lograr sus objetivos de diseño.

- *Recursos:* que pueden estar gestionados por agentes o por algún sistema propietario y que constituyen parte del entorno de los agentes.

Un recurso puede ser, por ejemplo, un servidor Web, una base de datos, un servidor de correo, un sistema de gestión de procesos de negocio, etc., dependiendo del entorno de la aplicación.

- *Un middleware o plataformas de agentes:* para facilitar la comunicación entre los agentes y con sistemas propietarios. Al referirnos a middleware incluimos servicios asociados (como la localización, las notificaciones y la seguridad).

Sobre este middleware es posible utilizar herramientas basadas en estándares de comunicación entre agentes como FIPA ACL o KQML[3].

Modelar con agentes tiene ventajas en el sentido de que este paradigma es bastante más cercano a nuestra manera de ver el mundo: muchas organizaciones se definen con un conjunto de roles y relaciones entre ellos, que luego son asumidos por personas (agentes).

Es pertinente decir que la problemática de este paradigma es que trata de recrear o simular como funcionan las interacciones interpersonales del mundo real (compra y venta de productos, búsqueda de información y otras actividades que involucran comunicación e interacción), lo cual es muy difícil de modelar ya que los seres humanos podemos ser vistos como sistemas dinámicos altamente complejos.

[3] ACL y KQML son estándares para creados para la comunicación entre agentes.

Aún así, es necesario disponer de métodos y herramientas para el desarrollo de los sistemas basados en agentes, y ésta es una de las actividades a las que se ha dedicado mayor atención.

1.6 Experiencias en el uso de Sistemas Multiagente

Los trabajos de investigación en el área de agentes han sido muchos, desde el momento de su concepción como paradigma de computación distribuida, hasta nuestros días donde vemos grandes avances en las áreas de Simulación Basada en Agentes entre otras.

Las investigaciones en el campo de agentes han tenido como resultado la creación de diversos middlewares o plataformas de agentes (estáticos y móviles).

Una plataforma de agentes se puede definir como una estructura en software, que facilita la operación e interacción entre agentes bajo ciertas restricciones (de seguridad mayormente) y que además les permiten tomar ventaja de los servicios y facilidades de cada sistema (ver Figura 1.4) [18].

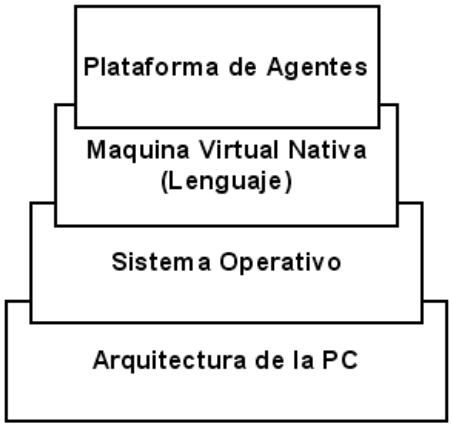


Fig. 1.4 – Nivel en donde opera una plataforma de agentes. Fuente: Los Autores.

En si, cada plataforma de agentes está fundamentada en el desarrollo e investigación de distintas actividades con agentes, por esto cada una presenta diferentes fortalezas como seguridad, escalabilidad, comportamiento de agentes, migración de agentes, etc. [36]

1.7 Problemas no Resueltos de los Sistemas Multiagentes

Los agentes móviles han sido llamados la solución al problema de diseñar e implementar aplicaciones distribuidas en ambientes dinámicos. Ellos proveen un grado de abstracción bastante simple de comprender e implementar.

Los agentes móviles son la forma más poderosa de *código móvil*[4], desafortunadamente mientras que otras formas de código móvil como el *Código Bajo Demanda y la Evaluación Remota* son ampliamente usadas, los agentes móviles no han sido bien recibidos por el público general.

Hay muchos problemas que deben ser solucionados para que exista confianza en las aplicaciones basadas en agentes móviles. Según Giovanni Vigna [44], existen diez problemas principales:

1. *El transporte de los agentes móviles es costoso*: los agentes móviles presentan un rendimiento desfavorable cuando se le compara con otras formas de código móvil, como evaluación remota e invocación de procedimientos remotos.

2. *Los agentes móviles son difíciles de diseñar:* la mayoría de las aplicaciones distribuidas son complejas y esta complejidad es mucho más fácil resolverla con los bien conocidos conceptos de modularización e interacción entre componentes. Con los diseños

[4] Código móvil se refiere a programas que se obtienen de sistemas remotos, transferidos por medio de la red y ejecutados localmente sin necesidad de instalación o ejecución por parte del usuario. Los tres paradigmas fundamentales del código móvil son: el código bajo demanda, la evaluación remota y los agentes móviles.

basados en sistemas de agentes es difícil identificar cuando estos interactuarán y como modelar estas interacciones.

3. *Los agentes móviles son difíciles de desarrollar*: el código del agente debe ser adaptativo, ya que este puede ejecutarse en condiciones impredecibles y ser capaz de comunicares con otros agentes o alguna plataforma nunca vista.

4. *Los agentes móviles son difíciles de probar (debug):* hacer pruebas de un software es considerado un arte, la movilidad agrega un grado de complejidad al proceso. Los agentes móviles tienen la capacidad de moverse de un nodo a otro en un orden no predefinido; ser capaz de hacer pruebas en un ambiente de ejecución tan complejo es increíblemente difícil.

5. *Los agentes móviles son difíciles de controlar:* los agentes móviles deben ser identificados cuando entran a un nuevo ambiente, esto requiere el uso de mecanismo basados en identidad. El problema es que existen muchas identidades relacionadas al agente, entre ellas: la identidad del que desarrollo el agente, la identidad de la plataforma que lo creó y la identidad de la plataforma del último nodo que visitó, y no está claro cuál de estas identidades es la que se debe usar para identificar inequívocamente al agente.

6. *Los agentes móviles pueden ser convencidos (brainwashed):* agentes que viajan a través de muchos nodos son vulnerables a ataques por parte de ambientes de ejecución maliciosos (plataformas maliciosas). Es posible que la plataforma modifique el código o el estado de ejecución y por lo tanto cambiar la manera que el agente se comporta.

7. *Los agentes móviles no pueden mantener los secretos:* los agentes se han propuesto como una solución en transacciones de comercio

electrónico, lo cierto que para que sean implementados en este tipo de infraestructura es necesario un mecanismo de seguridad como el cifrado o el uso de llaves privadas, de modo que solo la plataforma que tiene la función de descifrado o la llave privada correspondiente puede ver la información secreta.

8. *Los agentes móviles carecen de una infraestructura ubicua (una infraestructura que este presente en todas las plataformas de agentes):* como se mencionó anteriormente en este capítulo para que exista la movilidad de un agente es necesario un mecanismo de serialización efectivo capaz de representar fielmente el estado del agente. Este mecanismo no es el mismo en todas las plataformas y por lo tanto se convierte un problema a la hora de implementar agentes móviles en ambientes de varias plataformas.

9. *Los agentes móviles carecen de un lenguaje común (ontología):* los agentes móviles necesitan interactuar con el ambiente al que llegan para cumplir sus tareas. Esta interacción requiere que exista un lenguaje común entre el agente y la plataforma que lo recibe. Aunque se han propuesto varias lenguajes y ontologías, la realidad es que ninguna ha sido completamente aceptada.

10. *Los agentes móviles tienen un comportamiento muy similar a los gusanos (Worms):* el mecanismo de ejecución de los agentes móviles es muy parecido a la manera que se propagan los gusanos a través de una red. Una infraestructura de agentes móviles puede soportar la ejecución de agentes benignos y malignos, y por lo tanto es posible la ejecución de ataques como los que hacen los gusanos.

Es posible alegar que las razones dadas por Vigna no son del todo ciertas, de hecho a la fecha podemos decir que muchos de estos problemas como las ontologías en común, los mecanismos para mantener secretos utilizando

cifrado de llave pública y privada, al igual que la arquitectura ubicua están sufriendo avances significativos como se discutirá más adelante.

Desafortunadamente la realidad es que la diversificación en la creación de plataformas de agentes, la complejidad de los sistemas basados en agentes y la inexistencia de estándar único de construcción de agentes han causado grandes fallas en la tecnología de agentes, razones por la cual no ha habido una aceptación por parte del público general. Estas fallas se pueden agrupar en tres problemas fundamentales: [30]

- *Problemas de Interoperabilidad:* estándares de interoperabilidad entre agentes como The OMG Mobile Agent System Interoperability Facility (MASIF) y los de *Foundation for Intelligent Physical Agents* (FIPA) no han sido totalmente aceptados y es muy difícil comunicar agentes en un ambiente multiagente, donde estos vienen de diferentes plataformas.

- *Problemas de administración:* cuando una plataforma de agentes móviles es usada para formar una red dinámica, es muy difícil seguir a cada uno de los agentes para saber si un agente no ha podido cumplir su misión o si ya la cumplió y le ha tomado mas tiempo volver a su agencia o si simplemente se perdió en la red.

- *Problemas de seguridad:* es quizás uno de los factores fundamentales por lo cual no se le ha aceptado en el mundo de las Tecnología de Información y Comunicación (IT), la solución no ha sido completamente encontrada, ya que no existe un modelo único de seguridad. Algunos de los problemas son ataques de agentes maliciosos a computadoras, de computadoras maliciosos a agentes y ataques entre agentes.

En el siguiente capítulo se explicará a fondo cada uno de estos tres problemas fundamentales de la tecnología de agentes y se explicará las soluciones que se les ha encontrado.

2.1 Retos de los Sistemas Multiagentes

Como se dijo en el capítulo 1, existen tres problemas fundamentales que se deben resolver para que la tecnología de agentes tenga la acogida por el público general.

En este capítulo vamos a describir el por qué son problemas para el avance de la tecnología. Es necesario decir que vamos a desarrollarlos en orden de prioridad ascendente para la investigación, es decir, que primero hablaremos del problema de administración (el cual está fuera del alcance de la investigación), luego hablaremos sobre el problema de seguridad (problema que se puede solucionar utilizando una plataforma de agentes con medidas de seguridad) y por último el problema de interoperabilidad (problema central de nuestra investigación).

2.1.1 Problemas de Administración

Entre las razones expuestas por Vigna [44] encontramos cuatro que caen en el rubro de problemas de administración. A grandes rasgos podemos mencionar que los agentes son difíciles de desarrollar, diseñar y probar, por lo tanto difíciles de administrar.

El problema de administración no es visible cuando existe un número reducido de agentes y de nodos (a los cuales pueden llegar), pero sólo el hecho de imaginar 100 agentes moviéndose entre 100 nodos es bastante complicado. La plataforma de agentes tendría que mantenerse pendiente de las acciones de estos 100 agentes, sabiendo que algunos de ellos serán tomados como virus en algunas plataformas (y por lo tanto eliminados); algunos nodos pueden ser apagados antes de que los agentes puedan completar su migración a otro nodo (y por lo tanto perderemos la información que los agentes pudieron haber recolectado); algunos otros serán atacados por otros agentes o plataformas

maliciosas y solo la mitad (o quizás menos de la mitad) lleguen a completar sus tareas con éxito.

Aunque como hemos dicho el resolver esta situación está fuera del alcance de nuestra investigación pensamos que una solución parcial a este problema es el paso de mensajes por parte del agente a su respectiva plataforma (o agencia que lo creó), a la hora de llegar o partir a un nodo.

Los problemas con esta solución son: que se implica que todos los nodos (el de llegada y el de partida) se mantengan en línea mientras se hace el paso de mensajes, la migración y el paso de mensajes subsiguiente; también implica que ambos nodos (el de llegada y el de partida) tengan mecanismos de paso de mensajes conocidos para el agente. También estos mensajes son un problema, ya que representan un aumento en el tráfico de la red.

Ambas partes de nuestra solución son independientes al agente y por lo tanto incontrolables.

2.1.2 Problemas de Seguridad

A continuación se presentará, brevemente, los problemas se seguridad identificados en los Sistemas Multi Agentes (SMA).

La seguridad es considerada el mayor punto de controversia en la aplicación de soluciones basadas en sistemas de agentes [1].

Antes de exponer los problemas de seguridad relacionados con los SMA debemos tener presente cual es el fin de la seguridad en la tecnología de información: "Permitir que una organización logre todos sus objetivos de negocios y su misión mediante la implementación de sistemas con la consideración necesaria hacia los riesgos relacionados de IT para la organización, socios y clientes" [33].

Los SMA sufren de la mayoría, si es que no de todos, los problemas de seguridad de los sistemas actuales y además introducen muchos otros. En las 10 razones expuestas por Vigna [44], seis están relacionadas con seguridad. Sólo en el 2005 se reportaron 2368 vulnerabilidades de severidad media y alta en los sistemas actuales [32]. Esto representa un aumento del 107% por ciento con respecto al 2004. Estas cifras siguen aumentando cada año. Además las técnicas de seguridad de hoy en día están lejos de proveer soluciones definitivas a los problemas de seguridad. [45].

Se podría argumentar que si se diseña un SMA para ser totalmente seguro se pueden resolver todos los problemas de seguridad, pero plataformas como SeMoA[5] que han sido diseñadas para ser seguras, también tienen sus limitaciones.

Así como existen muchos problemas, también existen variedad de soluciones propuestas como: análisis estático y dinámico de los patrones de ataque [45], políticas de seguridad dinámicas durante la ejecución [21], agentes que se auto protegen [42] y otras [17].

Las amenazas se pueden clasificar de diversas maneras, en este trabajo utilizaremos la clasificación basada en los componentes de los SMA [31].

Hay cuatro categorías de amenazas identificadas: un agente atacando una plataforma, una plataforma atacando a un agente, un agente atacando a otro agente en la plataforma y otras entidades atacando al sistema de agentes.

[5] SeMoA. (Security of Mobile Agents). Es una plataforma de agentes que tiene como meta ser un ambiente seguro para la ejecución de agentes.

2.1.2.1 Agente a Plataforma

Es el conjunto de amenazas en las cuales el agente trata de explotar vulnerabilidades en la plataforma o lanza ataques directamente a ella.

- **Enmascaramiento:** Ocurre cuando un agente no autorizado se hace pasar por otro agente. El agente enmascarado podría solicitar servicios y recursos a los que no está autorizado. También puede hacerse pasar por otro agente no autorizado y así, hacer responsable a éste de sus acciones.

- **Denegación de servicio:** Es un ataque a un sistema que causa la pérdida o interrumpe el servicio a los usuarios. Un agente malicioso puede consumir de forma excesiva los recursos de una plataforma causando una interrupción del servicio. Este tipo de ataque puede ser iniciado intencionalmente por scripts que buscan explotar vulnerabilidades o por errores de programación. Los errores de programación son de particular importancia debido a que el paradigma de agentes móviles requiere la ejecución de código externo.

- **Acceso no autorizado:** La identidad de un agente móvil debe ser autenticada antes de ser instanciado en la plataforma. Una plataforma que brinda servicios a agentes de diversos usuarios y organizaciones debe garantizar los permisos (como los de lectura y escritura) de la información, incluyendo los permisos de la información residual almacenada en la memoria caché o cualquier otro tipo de almacenaje temporal.

2.1.2.2 Agente a Agente

Son el conjunto de amenazas en las cuales los agentes intentan explotar debilidades en otros agentes. Algunas plataformas permiten la comunicación

entre agentes de manera directa; otras requieren que todos los mensajes entrantes y salientes pasen por medio de un agente de comunicación de plataformas.

- **Enmascaramiento:** Un agente puede intentar disfrazar su identidad para engañar al agente con el cual se comunica. Este ataque causa daños a ambos agentes: el agente que fue engañado y el agente cuya identidad fue falsificada, en especial en agencias en donde la reputación es utilizada como una forma de establecer confianza entre agentes.

- **Denegación de Servicio:** Un agente puede inundar a otro enviándole un flujo constante de mensajes causando una carga en las rutinas de manejo de mensajes en el receptor. El agente que está siendo atacado puede bloquear los mensajes provenientes del agente malicioso, pero esto también requiere ciclos de CPU. Esto es un problema si al agente se le limita a cierta cantidad de tiempo del CPU.

  Agentes maliciosos también pueden divulgar información falsa para prevenir que otros agentes no puedan cumplir con sus tareas.

- **Repudio:** Una plataforma no puede evitar que un agente repudie una transacción, pero si puede garantizar la suficiente evidencia para ayudar a la resolución de desacuerdos. Los desacuerdos no sólo pueden surgir por el repudio de una transacción sino también por ciertos procesos de negociación que conducen a diferentes percepciones de un evento.

- **Acceso no autorizado:** Si una plataforma tiene mecanismos de control débil o inexistente, un agente puede invocar los métodos públicos, acceder o modificar el código o la data de otro agente.

2.1.2.3 Plataforma a Agente

Representan el conjunto de amenazas en las cuales la plataforma puede comprometer la seguridad los agentes.

Enmascaramiento: Una plataforma maliciosa puede hacerse pasar por otra plataforma para tratar de engañar al agente.

Denegación de Servicio: Una plataforma maliciosa puede voluntariamente ignorar las peticiones de servicio, introducir demoras en tareas críticas, no ejecutar el código del agente o terminar al agente sin notificación.

Espionaje ("Eavesdropping"): La plataformas pueden monitorear toda instrucción ejecutada por el agente, toda la información no cifrada o pública. A parte de que también puede interceptar las comunicaciones.

Alteración: El código, estado y datos del agente pueden ser modificados por una plataforma maliciosa. Esta amenaza se ve amplificada cuando un agente tiene una gran cantidad de agencias en su itinerario, sin importar que estas sean confiables ya que pueden surgir ataques de enmascaramiento para luego alterar al agente.

La alteración es un tema bastante complicado. Si no se detecta inmediatamente las modificaciones por una plataforma maliciosa, puede ser imposible descubrirla debido a que el agente puede realizar muchas visitas y sufrir cambios en todas ellas.

2.1.2.4 Otras entidades a la Plataforma

Esta categoría representa al conjunto de amenazas en la cual entidades externas, incluyendo agentes y plataformas, amenazan la seguridad de la plataforma.

- **Enmascaramiento:** Un agente en una plataforma remota puede enmascararse y solicitar servicios y recursos a los cuales no está autorizado. Los agentes maliciosos pueden trabajar en conjunto con una plataforma maliciosa.

  Las plataformas también se pueden enmascarar como otra plataforma y engañar a otras plataformas o agentes.

- **Acceso no autorizado:** Los accesos remotos a la plataforma y al equipo anfitrión deben ser protegidos cuidadosamente para evitar los ataques convencionales al sistema operativo o a servicios del mismo equipo. Permitir la administración remota facilita el manejo de las plataformas, pero también convierte la cuenta de administración en el blanco predilecto de muchos ataques.

- **Denegación de Servicio:** Los servicios ofrecidos por la plataforma pueden ser interrumpidos por ataques convencionales de DoS (ataques que causan la degradación o interrupción de un servicio a los usuarios). Las plataformas también son susceptibles a los ataques de DoS dirigidos al sistema operativo donde opera.

- **Copia y repetición:** Todo agente que se mueve de una plataforma a otra, al igual que sus mensajes, pueden ser capturados, modificados para luego ser retransmitidos.

2.1.3 Problemas de Interoperabilidad

Los agentes fueron creados para interactuar ya sea con otros agentes o con otros sistemas. La idea central del paradigma de agentes es que el trabajo de una comunidad de agentes tiene mejores resultados que los que puede tener agente trabajando solo, lo que implica la necesidad de interoperabilidad entre sistemas de agentes. [24]

Existen dos enfoques acerca de la interoperabilidad en sistemas de agentes son:

- Los que estudian los agentes móviles se interesan en los ambientes de ejecución y en la estandarización de algunos de sus aspectos y capacidades.

- Los que estudian agentes estáticos, no se preocupan por los ambientes de ejecución y su interés principal esta en la comunicación entre agentes, en otras palabras en el intercambio de la información y del conocimiento que conforman el contenido de los agentes.

Para los fines de esta investigación aceptamos como la definición más acertada de interoperabilidad: *La habilidad de una plataforma de agentes para ejecutar y dar soporte a cualquier agente de otra plataforma, de la misma manera que lo haría su plataforma original, sin necesidad de modificaciones en su estructura.*

En otras palabras, se dice que dos plataformas de agentes móviles son interoperables si un agente de una ***plataforma A*** puede migrar (mover su estado de ejecución) a una segunda ***plataforma B*** y ser capaz de interactuar con los agentes de esta plataforma (o alguna otra plataforma C) (ver figura 2.1).

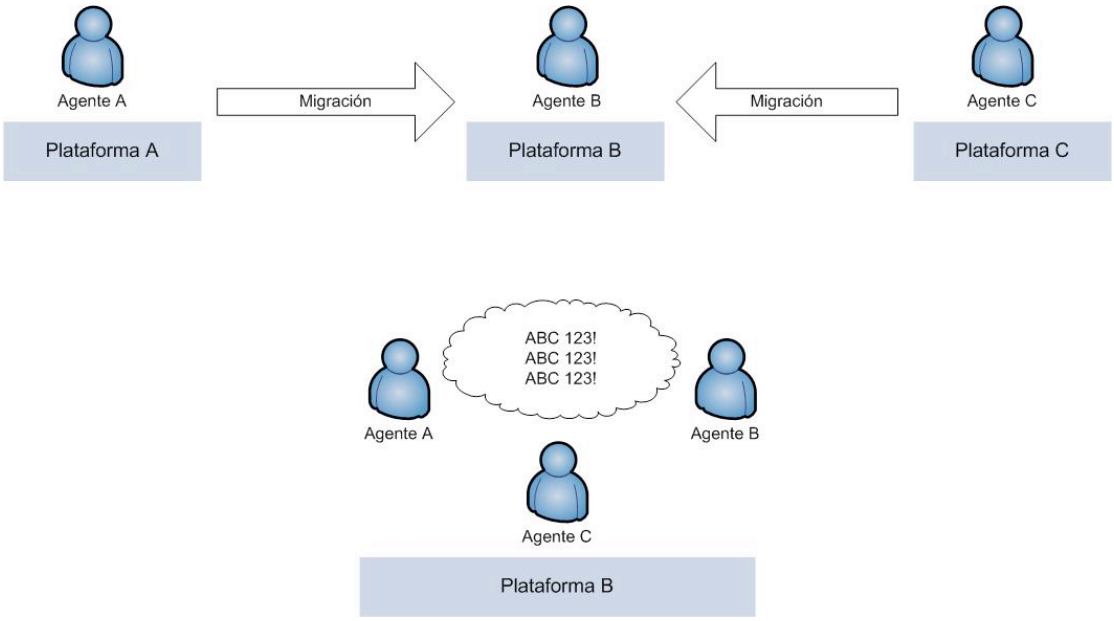


Fig 2.1. Los agentes A y C están en la plataforma B, continúan su ejecución e interactúan con otros agentes mediante un lenguaje de comunicación común entre ellos. Fuente: Los Autores.

El ámbito de trabajo de los agentes móviles por lo general son ambientes altamente heterogéneos. Los agentes migran a nodos donde un ambiente de ejecución está preparado para ellos; luego de su llegada ellos son capaces de ejecutar código y hacer llamadas a procedimientos remotos (RPC) con el fin de acceder a los recursos del nodo, colectar información y eventualmente iniciar el proceso de migración hacia algún otro nodo. Durante su estancia en un nodo es posible que el agente se comunique con los demás agentes que estén en el nodo mediante algún tipo de mecanismo de comunicación brindado por el nodo (entre los mecanismos más utilizados están los tableros o whiteboards y los mensajes entre agentes).

El gran problema de la interoperabilidad surge del hecho que todas las plataformas de agentes son diferentes y es posible que el agente no encuentre la manera de llevar a cabo su tarea en el nodo, hasta es posible pensar que una vez llegado a la plataforma nunca encuentre la manera de migrar de él.

Para solventar estos problemas se han creado estándares de interoperabilidad para las plataformas de agentes móviles, que buscan unificar la manera en que funcionan los ambientes de ejecución en diferentes plataformas. La realidad es que algunos de estos estándares han caído en desuso y en la mayoría de los casos su utilización es decisión del creador de la plataforma y por lo tanto han salido nuevas plataformas, algunas siguen los estándares y otras NO.

2.2 Estándares para plataformas de agentes

2.2.1 Estándares OMG MASIF y ARPA KSE

El Object Management Group (OMG) es una asociación que contiene más de 800 empresas e instituciones, fue formada en 1989 y tiene como objetivos: la interoperabilidad, reusabilidad y la portabilidad de componentes de software orientado a objetos en sistemas distribuidos. Esta asociación fue la responsable de la creación de estándares ampliamente utilizados como: Common Object Request Broker Architecture (CORBA)[6] y el Unified Modeling Language (UML)[7] . [27]

En 1995 OMG lanza OMG MASIF (Mobile Agent System Interoperability Facility) el cual fue un intento por parte del OMG para estandarizar algunos aspectos de los ambientes de ejecución de las plataformas de agentes. OMG MASIF estaba constituido por una colección de definiciones y especificaciones que proveían una interfase interoperable para los sistemas de agentes móviles.

La propuesta de interoperabilidad de OMG MASIF buscaba la interoperabilidad entre plataformas de agentes escritas en el mismo lenguaje de programación (por ejemplo JAVA), aunque que no existiese un consenso entre los creadores de las mismas.

[6] CORBA es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.
[7] UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software

OMG MASIF se enfocaba en estandarizar 4 partes primordiales:

- *Administración de Agentes:* operaciones normales como crear un agente, suspender, resumir y terminar su ejecución.

- *Transferencia de Agentes:* una infraestructura común para que los agentes pudieran moverse libremente entre sistemas de agentes distintos.

- *Nombres para los Agentes y los Sistemas de Agentes:* la utilización de una sintaxis y semántica estándar a lo hora de nombrar los agentes y los sistemas de donde provienen, de modo que sea posible identificar a los agentes y sus sistemas relacionados.

- *Seguimiento de Agentes:* presenta un mecanismo para saber dónde están los agentes y un directorio para saber sus funciones.

OMG MASIF no incluía:

- *Seguridad a la hora de migrar hacia otro nodo:* no presentaba un mecanismo estándar de seguridad en la migración y se asumía ambientes sin malicia.

- *Movimiento de los agentes entre plataformas de agentes heterogéneas:* conversión en el código del agente al de la plataforma donde llega.

- *Comunicación de Agentes:* se dejó la decisión a los creadores de plataformas, no se definió ningún lenguaje de comunicación, ni ningún protocolo de negociación.

Las interfaces de OMG MASIF se definían a nivel de la plataforma de agentes, sin tomar en cuenta los agentes. Las plataformas de agentes y los agentes podían ser objetos CORBA, mas no era una obligación.

Al final podemos decir que en OMG MASIF no existía una interfaz unificada para los agentes, ya que los agentes sólo viajaban a sistemas de agentes que soportaran el perfil OMG MASIF.

ARPA KSE

Knowledge Sharing Effort o KSE fue iniciado por Advanced Research Projects Agency (ARPA) hacia 1990, y fue apoyado por organismos norteamericanos de investigación como Air Force Office of Scientific Research (ASOFR), National Science Foundation (NSF), Natural Resources Institute (NRI). [27]

Tenía como propósito el desarrollo de técnicas, metodologías y herramientas software para compartir y reutilizar el conocimiento entre sistemas a lo largo de las etapas del ciclo de vida del software: diseño, implementación y ejecución.

Con el tiempo sus creadores se dieron cuenta que para compartir conocimientos entre agentes se necesitaba que los agentes tuvieran la capacidad de comunicarse mediante un lenguaje de comunicación común, por lo que crearon los estándares: KIF (Knowledge Interchange Format) para que se encargara de la Sintaxis, Ontolingua (lenguaje para definir ontologías) para que se encargara de la Semántica y KQML (Knowledge Query and Manipulation Language) para que se encargara de la parte pragmática, en otras palabras lenguaje de comunicación en sí.

De estos tres estándares, hablaremos de KQML ya que es el predecesor más cercano al lenguaje de comunicación ACL, del cual hablaremos luego en este capítulo.

KQML fue creado para ser un lenguaje de comunicación y protocolo, orientado a mensajes, que propiciara el intercambio de información, independiente de protocolos de transporte (TCP/IP, HTTP), tener sintaxis de contexto libre, soportar ontologías y protocolos de alto nivel (contract net, subasta).

KQML asume un modelo de agentes como entidades de alto nivel con ciertas características, entre ellas:

- Poseen capacidades cognitivas (capacidades como representación simbólica y utilización de una base de conocimientos).

- Poseen una descripción de nivel intencional (para plataformas del tipo BDI): su estado es un conjunto de componentes mentales como Creencias, Capacidades, Elecciones y Compromisos.

Los mensajes KQML comunican una acción en el contenido que llevan, ésta puede ser una solicitud o una pregunta.

Las primitivas del lenguaje KQML se les llama preformativas, que se basaban en el concepto de la *Teoría del Acto del Habla* (que sugiere que toda frase comunicada es implícitamente o explícitamente una acción).

Aunque KQML se ha dejado de utilizar como lenguaje de comunicación entre agentes es necesario decir que, actualmente existen varios dialectos de KQML con distintas extensiones añadiendo preformativas, añadiendo nuevos parámetros y añadiendo aspectos de seguridad.

2.2.2. FIPA

2.2.2.1. Historia y Características de FIPA

Como ocurre con toda nueva tecnología, uno de los problemas que hay que resolver para facilitar su aplicabilidad es la interoperabilidad (o facilidad de interconexión e integración de sistemas basados en dicha tecnología) y la apertura (posibilidad de extensión). Para ello es importante disponer de estándares. [13]

En el caso de los agentes, una de las organizaciones que más ha trabajado en este sentido es FIPA (Foundation for Intelligent Physical Agents), que es un consorcio industrial fundado en 1996, conformado por decenas de compañías de telecomunicaciones e informática. Entre sus objetivos está el acelerar el desarrollo de tecnologías de agentes inteligentes mediante la producción de especificaciones acordadas internacionalmente.

Las investigaciones de FIPA abarcan todo tipo de agentes (estáticos, móviles), sistemas multiagentes y sociedades de agentes.

Las especificaciones de FIPA están basadas en casos prácticos concretos. De hecho, existen varias especificaciones dentro de los estándares de FIPA, ellas son FIPA 97, FIPA 98 y FIPA 2000.

En las especificaciones FIPA 97 y FIPA 98, se trató de estandarizar o ayudar a estandarizar ciertos aspectos de la tecnología de agentes como lo son:

- Gestión de agentes
- Lenguaje de comunicación entre agentes
- Interacción agente-humano e Interacción de agentes con otros sistemas
- Movilidad de agentes
- Seguridad de agentes

También se sugirieron ciertas aplicaciones de prueba para el uso de los agentes como:

- Agente personal de viaje
- Asistente personal difusión y entretenimiento audio-visual
- Gestión de redes

La especificación FIPA 2000 presenta una evolución en el modelo de comunicaciones de agentes, en el transporte de los mensajes, representación de los mensajes (como cadenas de caracteres, objetos, o XML) y se agrega atributos opcionales en los mensajes (autenticación, cifrado). También se define la arquitectura abstracta por medio de la cual dos agentes se pueden localizar y comunicar.

Para FIPA los agentes se comunican intercambiando mensajes que representan actos de habla codificados en un lenguaje de comunicación de agentes.

Los servicios dentro de una plataforma se pueden implementar como agentes o como software que se accede invocando métodos. El estándar FIPA define ciertos servicios que debe proporcionar toda plataforma de agentes:

- Un sistema encargado del transporte de mensajes (Internal Platform Message Transport)
- Un sistema de gestión de agentes (Agent Management System)
- Un servicio facilitador de directorio (Directory Facilitator)
- Un canal de comunicaciones para los agentes (Agent Communication Channel).

En la figura 2.2 se puede ver el Modelo de Referencia de FIPA, es de notar que es el Sistemas de Transporte de Mensajes el que interactúa con otras plataformas de agentes.

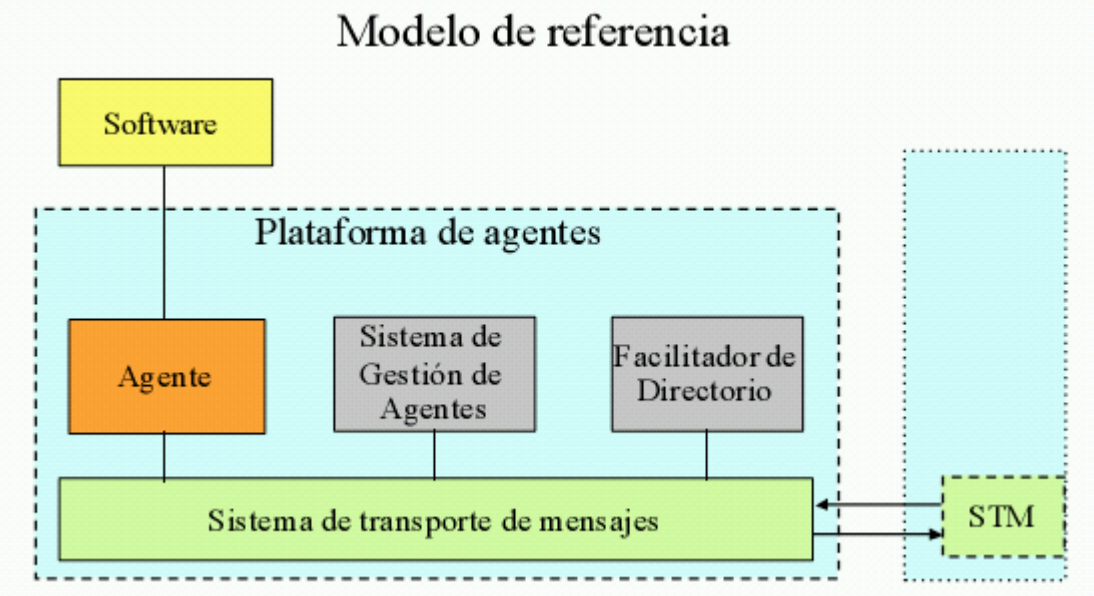


Fig 2.2 – El modelo de referencia para una plataforma de agentes propuesto por FIPA. Fuente: Corchado, J. Modelos y Arquitecturas de Agentes. Universidad de Salamanca.

Cada uno de estos servicios, excepto el de transporte de mensajes, es suministrado por agentes especializados, lo cual supone que la comunicación con ellos será mediante mensajes ACL mediante la ontología definida para ese servicio.

Cuando la especificación hable de mensajes se refiere a cadenas con el formato *clave:valor* escritos en un lenguaje de comunicación de agentes (FIPA ACL) y que su contenido es expresado con un lenguaje de contenidos (KIF) o puede hacer referencia a una ontología (vocabulario).

A la hora de definir la plataforma de agentes, FIPA ha seguido el principio de definir sólo el comportamiento externo (interfaz), dejando las decisiones de diseño a cada equipo de desarrollo. Otro principio que guía todo el estándar es conseguir un sistema totalmente abierto de tal forma que sistemas heterogéneos puedan interactuar a nivel de sociedades de agentes. El modelo FIPA establece el modelo lógico referente a la creación, destrucción, registro, localización y comunicación de agentes.

La Plataforma de Agentes (Agent Platform), como núcleo del modelo de referencia de FIPA, proporciona la infraestructura para el desarrollo y uso de agentes. Esta plataforma contiene todos los recursos hardware y software (sistema operativo, software de comunicaciones, middleware y software de gestión de agentes) necesarios para poner en marcha esta infraestructura.

La idea central de los estándares de FIPA es que en cada implementación concreta se tomen las decisiones relativas a las componentes usadas para desarrollarla, pero manteniendo siempre un conjunto de interfaces externas que permitan la interoperabilidad entre plataformas (por ejemplo para comunicación y gestión de agentes).

2.2.2.2 Estándares de FIPA

FIPA ACL

Se basa en la teoría de actos del habla, por lo tanto los mensajes son acciones comunicativas, su semántica se basa en aptitudes mentales (Creencias, Deseos e Intenciones), tiene una sintaxis similar a KQML y provee semántica formal definida con lógica modal[8].

También se definen protocolos de interacción de alto nivel, llamados conversaciones. Es posible definir nuevas primitivas a partir de un núcleo de primitivas mediante composición.

Seguridad en el desarrollo de Sistemas Multiagente

Los problemas de seguridad son inherentes a cualquier sistema distribuido y más aún en sistemas multiagentes, debido a las interacciones que pueden existir entre los distintos agentes. Por ello FIPA propone soluciones de

[8] La lógica modal se ocupa de proposiciones afectadas por posibilidades.

seguridad para que los sistemas multiagentes puedan operar con las garantías requeridas.

Las recomendaciones de FIPA están basadas en la utilización de los estándares existentes siempre que esto sea posible. Por ello se plantea el uso del estándar de seguridad X.509[9] basado en infraestructuras de seguridad de clave pública (PKI). El estándar X.509 presenta un modelo basado en certificados de clave pública y clave privada y su funcionamiento se puede ver en la figura 2.3.

Para cada entidad se definen dos tipos de claves, la clave privada a las que sólo tiene acceso la entidad propietaria del certificado, y la clave pública a la que tienen acceso todas las demás entidades de la sociedad. Por cada entidad, la autoridad certificadora correspondiente genera un par de claves (pública y privada) para el cifrado (confidencialidad) y otro par de claves (pública y privada) para la firma (autenticidad). Cada entidad recibe sus claves privadas por parte de la autoridad certificadora.

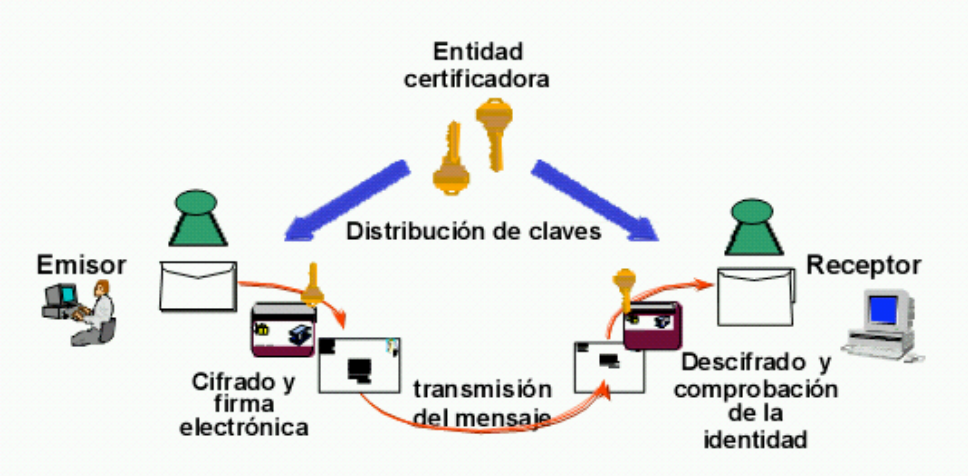


Fig 2.3 – Explicación de la Infraestructura de clave publica (PKI). Fuente: Corchado, J. Modelos y Arquitecturas de Agentes. Universidad de Salamanca

Las claves públicas de todas las entidades están guardadas en una estructura de directorio (LDAP)[10] y pueden ser consultadas en cualquier momento. También es posible que una entidad distribuya sus claves públicas a las entidades con las que quiere comunicarse de forma segura.

El uso de X.509 ofrece seguridad a nivel de agentes, pero es posible asegurar la conexión utilizando sockets seguros o SSL (Secure Socket Layer) [11], a nivel de la capa de transporte.

Movilidad de Agentes en la Plataforma

El estándar, FIPA no propone soluciones tecnológicas sino más bien aporta ideas acerca de cómo integrar de forma homogénea y natural las capacidades de los agentes móviles dentro de la plataforma. La interoperabilidad entre sistemas es muy importante en este campo, pues si dos sistemas de distinto origen no pudiesen intercambiar agentes la movilidad no sería aprovechable.

FIPA define dos modelos de movilidad en función de quién tenga que hacer la gestión de la transferencia:

- *Modelo Simple:* El AMS (Agent Management System) es el responsable de realizar toda la gestión necesaria. El agente solicita a su AMS la transferencia y éste se ocupa de llevarla a cabo.

- *Modelo Complejo:* El agente es el responsable de realizar todas las operaciones que le permiten desplazarse a la plataforma remota.

[9] X.509 es un estándar para infraestructura de claves pública (public key infrastructure o PKI). X.509 especifica, el formato estándar para certificados de claves públicas y un algoritmo de validación de la certificación.
[10] LDAP (Lightweight Directory Access Protocol) es un protocolo de red que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red.
[11] SSL es un protocolo criptográfico que proporciona autentificación y privacidad de la información entre extremos sobre Internet.

Transporte de mensajes de la plataforma de agentes

El APMT (Agent Platform Message Transport) es el encargado de realizar la transferencia de mensajes ACL entre las ACCs (Agent Communication Channel) de dos plataformas. Aunque en un principio la labor de diseño de este componente de bajo nivel se dejó en manos de cada grupo de trabajo que implementase la plataforma que sigue las especificaciones de FIPA, finalmente se decidió introducir dentro del propio estándar debido a la gran importancia que tiene dentro del sistema.

En la actualidad la mayoría de los entornos de desarrollo y ejecución de agentes tienden a adoptar, o ser compatibles, con FIPA, para tener su respaldo y ser capaces de interoperar con las otras plataformas que siguen el estándar.

2.2.3. IEEE FIPA

Cuando consideramos el grupo de estándares que han sido creados por FIPA, podemos darnos cuenta que han cumplido con su cometido. Pero, es tiempo de llevar los estándares de agentes y de sistemas de agentes a un contexto de desarrollo de software general, en otras palabras la tecnología de agentes necesita interactuar con tecnologías no basadas en agentes.

Con esta perspectiva en mente Computer Society del IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) ha invitado a FIPA para formar parte de su familia de comité de estándares.

En marzo de 2005, la junta directiva y los miembros de FIPA votaron unánimemente para unirse a la Computer Society del IEEE. Desde el momento de la unión a IEEE ahora se reconoce a FIPA como el 11avo comité de estándares de la IEEE, bajo el nombre de *FIPA Standards Comitee (FIPA SC)* [19].

La Computer Society del IEEE le provee al FIPA SC un paraguas organizacional y todos los beneficios que reciben de los demás comités de estándares actualmente funcionales en el IEEE.

El Enfoque del FIPA SC es que FIPA cumplió con su función en el tiempo que existió pero es necesario adaptarse a los tiempos actuales y reinventar la organización dentro del contexto de IEEE. También promover la colaboración entre grupos de trabajo y evitar el enfoque del pasado *de crear el estándar y esperar que todo mundo se ajuste a él.*

Algunas de las misiones del IEEE FIPA son:

- Promover los ambientes de integración para agentes y sistemas de agentes basados en estándares de la industria.
- Promover una infraestructura para el desarrollo de aplicaciones compatibles e independientes y promover su aceptación en el mercado y su utilización.
- Permitir la coordinación entre aplicaciones de sistemas heterogéneos interconectados.
- Trabajar con los demás grupos de estándares que conforman la FIPA SC mediante colaboración y la reutilización de estándares.

La FIPA SC tiene dos clases de grupos [20] que contribuyen en la creación de los estándares: los grupos de trabajo (WG) y los grupos de estudio (SG), debemos decir que los grupos de estudio no producen especificaciones para los estándares, ésta es la labor de los grupos de trabajo.

Los grupos de trabajo llevan a cabo proyectos de estandarización. Cada grupo de trabajo es responsable por el contenido de uno o más proyectos de estandarización.

Los grupos de trabajo que se han aprobado son los siguientes:

- *Grupo de trabajo sobre Interoperabilidad entre Agentes y Servicios Web (AWSI WG):* su principal interés es lograr la interoperabilidad entre agentes y servicios web y la utilización de la Web Semántica[12].

- *Grupo de trabajo sobre Comunicación Humano-Agente (HAC WG)*: su enfoque principal es la comunicación humano-agente en el contexto de toma de decisiones.

- *Grupo de trabajo de Agentes nómadas P2P (P2PNA WG):* tiene como fin definir las especificaciones referentes a agentes nómadas P2P, capaces de ejecutarse en dispositivos pequeños y la implementación de aplicaciones en dispositivos electrónicos de consumo como portátiles y robots, todo sobre redes P2P[13].

- *Grupo de trabajo sobre Agentes Móviles (MA WG):* tiene como fin definir nuevas especificaciones para: Movimiento de información. Seguimiento de agentes, infraestructura para el descubrimiento de plataformas de agentes e interoperabilidad entre diferentes plataformas de agentes móviles a nivel de ejecución.

Entre los planes del MA WG está generar tres (3) documentos que contendrán las especificaciones. Los documentos y su fecha de entrega son [8]:

- Especificación experimental de IEEE FIPA para Agentes Móviles, Junio de 2006

[12] La Web semántica (del inglés semantic web) es la idea de añadir metadatos semánticos a las paginas Web, osea información adicional como: descriptores de contenido, significado y relación entre los datos

[13] Una Red P2P o Peer-to-Peer es una red informática entre iguales, de modo que se refiere a una red que no tiene clientes y servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red.

- Referencia de Implementación, Octubre de 2006
- Especificación Estándar de IEEE FIPA para Agentes Móviles, Diciembre de 2006.

Es de importancia resaltar este último grupo (MA WG), pues es el que ha sido encargado de crear el próximo estándar para la interoperabilidad entre plataformas de agentes móviles, también es importante señalar que parte entre los miembros de esta comisión están: Ulrich Pinsdorf, uno de los creadores de la plataforma SeMoA (una de las plataformas a utilizar en nuestra investigación) y Peter Braun, uno de los creadores de la plataforma Tracy y autor del libro *Mobile Agents* [10].

2.3 JAVA, el Lenguaje de las plataformas de agentes

2.3.1 Características de JAVA útiles en las plataformas de agentes

El lenguaje JAVA es un lenguaje de uso general, usado mayormente en sistemas distribuidos y en los sistemas multiagentes, por que su diseño orientado a objetos y su manejo de componentes. [28]

Algunas de las ventajas del lenguaje JAVA, que lo hacen muy útil para ser utilizado en sistemas multiagente son:

- Soporta código móvil.
- Es un lenguaje con restricciones de seguridad.
- Capacidad de serialización de objetos (JAVA Serialize).
- Capacidad de introspección de los objetos, utilizando los métodos de JAVA Reflection[14].
- Cargado dinámico de clases.
- Multihilos de ejecución.

[14] Reflexión es la habilidad que tiene un programa en ejecución de examinarse a sí mismo y a su ambiente y cambiar dependiendo de lo que encuentra.

Algunas desventajas son:

- Poco soporte para el manejo de los recursos de disco y memoria.
- Poco soporte para preservar el estado de ejecución de un hilo (es imposible serializar un hilo).
- Soporte limitado para el manejo de versiones.

2.3.1.1 Protocolo RMI

RMI (Remote Method Invocation) habilita al programador para crear aplicaciones basadas en tecnología JAVA distribuida, en la cual los métodos de objetos remotos (posiblemente en diferentes servidores) pueden ser invocados como si fueran locales.

El sistema RMI consiste de cuatro capas:

- *Capa de aplicación*
- *Capa de fragmento/esqueleto:* transmite los datos de la capa de aplicación a la capa de referencia remota
- *Capa de referencia remota:* es responsable de proveer la habilidad para soportar variaciones de referencias remotas o invocaciones a protocolos independientes del fragmento del cliente y del esqueleto del servidor
- *Capa de transporte:* es responsable de establecer conexiones a direcciones remotas, manejar las conexiones, escuchar sus llamadas entrantes, manejar una tabla de objetos remotos que residen en el mismo espacio de direcciones, establecer una conexión de una llamada entrante localizando su despachador y pasándole su conexión.

Una de las ventajas al diseñar un procedimiento con RMI es interoperabilidad, ya que RMI forma parte de todo JSDK[15], por ende, cualquier plataforma que tenga acceso a un JSDK también tendrá acceso a estos procedimientos.

2.3.1.2 Características de Seguridad (JCE/JCA)

El API[16] de seguridad es parte del API principal del lenguaje JAVA, armado en el paquete *JAVA.security* y sus subpaquetes. Este API se diseñó para permitir a los programadores incorporar seguridad de alto y bajo nivel en sus programas.

La primera versión del API de seguridad en el JDK 1.1 introdujo la "Arquitectura de Criptografía de JAVA" (en inglés JAVA Cryptography Architecture - JCA), una plataforma para acceder y desarrollar funciones criptográficas sobre JAVA. En el JDK 1.1 se incluyeron API's para la firma digital y resúmenes (MD5).

En la versión 2 del JDK se mejora la JCA, incluyendo el manejo de certificados digitales (X.509 versión 3) y una nueva arquitectura de seguridad configurable y con controles flexibles para su implementación.

La "Extensión Criptografía de JAVA" (en inglés JAVA Cryptography Extension - JCE), amplía el API de la JCA incluyendo API's para cifrado, intercambio de llaves y "Código de Autenticación de Mensajes" (en inglés Message Authentication Code - MAC).

[15] JSDK (Java Software Development Kit) es un paquete ofrecido por Sun Microsystems con las utilidades y herramientas necesarias para el desarrollo de programas en el lenguaje JAVA.
[16] Una API (del inglés Application Programming Interface) es un conjunto de especificaciones de comunicación entre componentes de software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general.

Juntos la JCE y la JCA proveen un API criptográfico completo listo para usar e independiente de plataforma. JCE es parte de JAVA 2 SDK, desde la versión 1.4.

Principios de Diseño de la JCA

La JCA fue diseñada considerando estos principios:

- Independencia de la implementación e interoperabilidad: La independencia en la implementación se consigue mediante una arquitectura basada en proveedores. El proveedor de servicios criptográficos (en inglés Cryptographic Service Provider) es un grupo de paquetes que implementan uno ó más servicios criptográficos, entre ellos: algoritmos de firma digital, algoritmos de resumen de mensajes (en inglés message digest algorithms) y servicios de conversión entre llaves.

  Un programa puede solicitar un tipo específico de objeto (por ejemplo una objeto de firmado) implementando un servicio particular (por ejemplo el algoritmo DSA) y obtener su implementación gracias a alguno de los proveedores instalados. Los proveedores pueden ser actualizados cada vez que salgan algoritmos más rápidos o versiones más seguras de ellos.

  La interoperabilidad significa que varias implementaciones pueden trabajar juntas, intercambiar llaves públicas, verificar sus firmas digitales, etc.

- Independencia del algoritmo utilizado y capacidad de extensión: la independencia del algoritmo utilizado se logra mediante la definición de servicios y clases que le dan la funcionalidad a los servicios.

La capacidad de extensión significa que nuevos algoritmos que puedan ser soportados por los servicios pueden ser añadidos fácilmente.

La independencia de la implementación y la independencia del algoritmo utilizado son complementarias, es posible utilizar los servicios criptográficos (como firmas digitales, resúmenes de mensajes) sin preocuparse de los detalles de la implementación o de los algoritmos que son la base de estos conceptos.

2.3.1.3 Serialización

Serialización es el proceso de convertir un conjunto de instancias de objeto que contienen referencias entre ellas en un flujo lineal de datos
que puede ser almacenado, enviado o simplemente manipulado como un flujo de datos.

La serialización es comúnmente utilizada para salvar el estado de una aplicación como una representación binaria de todos sus objetos con sus valores correspondientes. Desafortunadamente los objetos serializados son frágiles, ya que si se hacen cambios en las clases que se usaron para producirlos, estos deben ser regenerados. Para solucionar esto los API´s basados en XML que manejan la serialización en JAVA (JSX - JAVA Serialization to XML) proveen una mejor alternativa, transformar los objetos en XML, que no es mas que texto legible y editable.

En JAVA la serialización es un mecanismo que se encuentra en las librerías principales del lenguaje y se usa para transformar un objeto en un flujo de datos (texto). El proceso tambien se puede hacer de forma inversa, es decir de un flujo de datos recrear un objeto, a esto se le llama deserialización.

Existen tres usos fundamentales de la serialización:

- *Como un mecanismo de alojamiento:* si el flujo utilizado en la serialización es FileOutputStream, entonces los datos se escribirán en un archivo.
- *Como un mecanismo de copiado:* si el flujo utilizado en la serialización es ByteArrayOutputStream, entonces los datos se escribirán en memoria. Este arreglo puede utilizarse para crear duplicados de los objetos originales.
- *Como un mecanismo de comunicación:* si el flujo utilizado en la serialización es un socket, entonces los datos serán automáticamente enviados por este canal de comunicación, donde otro programa decidirá que hacer con él.

Es importante notar que el uso de la serialización es independiente del algoritmo utilizado para serializar el flujo mismo.

2.4 Plataformas de Agentes (a utilizar en la investigación)

2.4.1 Plataforma JADE

2.4.1.1 Historia y Características

JADE (JAVA Agent DEvelopment Framework) es una plataforma implementada en el lenguaje JAVA y el requisito mínimo para su ejecución es la versión 1.4 del compilador de JAVA (puede ser el SDK o el JRE de J2SE). Simplifica la implementación de sistemas multiagentes mediante una plataforma de agentes que cumple con las especificaciones de FIPA y que contiene un grupo de herramientas útiles en las fases de eliminación errores y de implementación de sistemas multiagente. La plataforma de agentes puede estar distribuida en varios nodos (que pueden no estar ejecutando el mismo sistema operativo) y la configuración puede ser controlada remotamente

mediante una interfaz grafica. La configuración puede ser cambiada en tiempo de ejecución moviendo a los agentes de un nodo a otro, cuando sea necesario.

Telecom Italia Lab (Tilab) parte de Telecom Italia Group es la responsable de promover la innovación tecnológica y exploración de nuevas tecnologías, haciendo estudios de factibilidad y desarrollando prototipos y emuladores de los nuevos servicios y productos. Tilab concibió la idea y desarrolló JADE y creó una comunidad con la filosofía de código libre en febrero del año 2000.

2.4.1.2 Estructura de la plataforma JADE

JADE está compuesta de una plataforma FIPA para la ejecución de Agentes y un conjunto de paquetes para la programación de agentes, bajo las especificaciones de FIPA. [7]

JADE cumple con todas las especificaciones de los estándares FIPA, y por ello tiene ciertas características como:

- Soporta todos los métodos de codificación de mensajes FIPA y también soporta los lenguajes de manejo de contenido como XML, SL y RDF.

- Soporta el uso de múltiples contenedores, los cuales pueden estar ejecutándose en diferentes computadoras formando una plataforma de agentes.

- Soporta los servicios de Páginas blancas y amarillas, también permite la interconexión entre servicios definidos por el usuario y la subscripción a los mismos.

- La programación de los agentes se basa en el lenguaje JAVA y soporta comportamientos y eventos. También para programar es posible

utilizar JESS (Java Expert System Shell), que es un motor de inferencias basado en reglas.

- Soporta el uso de ontologías definidas y movilidad de agentes.

- Tiene herramientas de depuración como el visor de mensajes y el introspector de agentes.

- Soporta múltiples protocolos de comunicación y los protocolos adicionales se pueden adaptar mediante el protocolo de transferencia de mensajes (Message Transfer Protocol).

- La plataforma ofrece escalabilidad y una gran velocidad en las comunicaciones y Soporta diferentes máquinas virtuales como J2ME, J2EE y la plataforma .NET de Microsoft.

- Soporta adaptaciones de seguridad y servidores web.

- Existe un gran soporte, debido a la gran comunidad de desarrolladores que usan la plataforma JADE.

JADE se puede ejecutar en una o varias máquinas virtuales (JVM), cada máquina virtual es vista como un entorno en donde los agentes pueden ejecutarse concurrentemente e intercambiarse mensajes. En la figura 2.5 se puede ver una plataforma JADE distribuida con tres contenedores, un nodo principal y dos secundarios.

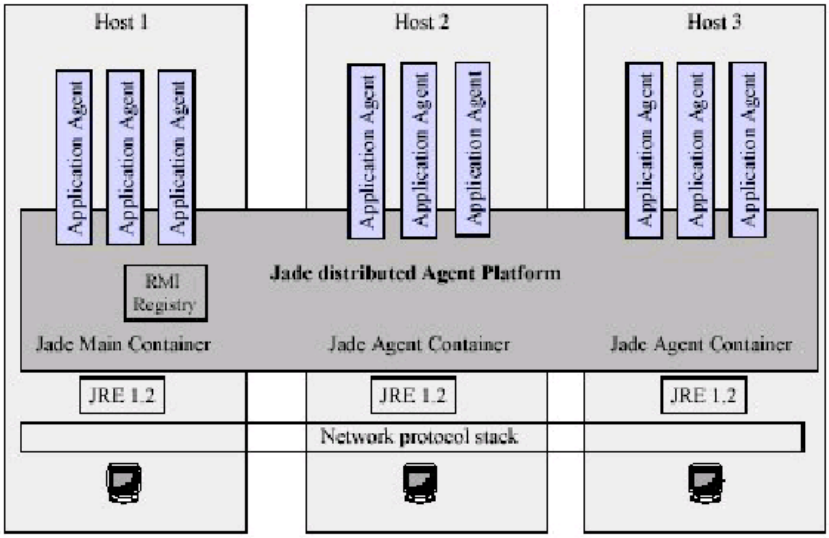


Fig. 2.5 – Cada máquina virtual (JRE) es vista como un entorno que forma parte de la plataforma de distribuida de JADE. Fuente: Botía J. La Plataforma de Agentes JADE.

La plataforma está organizada en uno o varios contenedores, donde uno funciona como contenedor principal que contendrá al agente servidor de mensajes (AMS), directorio de facilidades (DF) y el registro RMI, el resto de lo contenedores son *No principales* y funcionan conectados al principal.

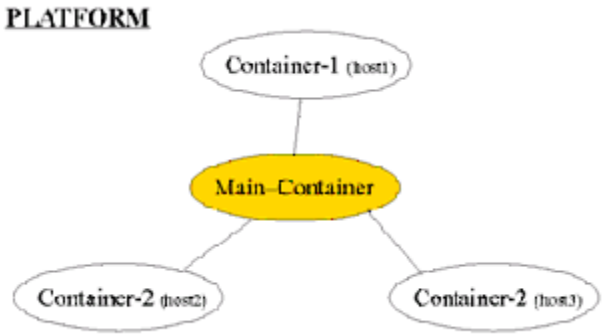


Fig. 2.6 - Los agentes jade operan dentro de un contenedor alojado en una plataforma de agentes, estas pueden estar distribuidas, conteniendo múltiples contenedores. Fuente: Botía J. La Plataforma de Agentes JADE.

Comunicación de agentes en JADE

Los mensajes son pasados por medio del agentes mensajero AMS entre agentes y permitiendo así su comunicación; los agentes comunicarse con otros agentes dentro de su mismo *contenedor* y también es posible el paso de mensajes a contenedores remotos.

El agente también puede migrar a un nuevo contenedor dentro de la misma plataforma y utilizar los métodos de paso de mensaje local, para minimizar el uso de ancho de banda.

Movilidad en Jade

A la hora de migrar o moverse de un contenedor a otro los agentes deben invocar (llamar) a estos métodos de la plataforma JADE:

- domove(): para migrar a un nodo deseado
- beforemove(): se utiliza para asegurar que el agente está preparado para migrar
- aftermove(): la ejecución continua una vez el agente haya completado la migración.

2.4.1.2.1 El Agente JADE

Programar un agente en JADE consiste en definir una clase JAVA que represente al agente y en ella determinar y codificar los comportamientos (reacciones) que va a manifestar el agente.

Los agentes JADE son identificados por un nombre único dentro de su contenedor, este se llama AID o JADE Agent ID. Los AID tienen la forma *<nombre-del-agente>@<nombre-de-plataforma>*.

La clase que representa al agente debe heredar los métodos y atributos de la clase JADE.core.Agent y reescribir sus métodos setup(), que es la clase utilizada como constructor (inicializador) de la clase y el método takeDown(), que es el método destructor (finalizador) de la ejecución.

Para ejecutar el agente se puede hacer desde la interfaz que nos ofrece la plataforma JADE o desde cualquier otro programa escrito en JAVA.

Comportamiento de un agente JADE

Los agentes deben poder ejecutar diferentes tareas al mismo tiempo. JADE propone un modelo de agente con único hilo de ejecución (por lo tanto monotarea) y añade una programación (scheduling) para esa tarea mediante comportamientos [12]. Para lograr la programación de los comportamientos es necesario:

1- Determinar que debe ser capaz de hacer el agente
2- Asociar cada funcionalidad con un comportamiento
3- Escoger el tipo de comportamiento de los tres (3) posibles.

   Comportamiento Una Sola Vez (One shot): la reacción solo se produce una vez.
   Comportamiento Cíclico: la reacción se repite cada vez que se llamada.
   Comportamiento Genérico: generan diferentes reacciones dependiendo del estatus del agente.

4- Dejar a la plataforma JADE se encargue de verificar que un solo comportamiento se ejecute en cada instante.

Programación de comportamientos (scheduling)

Las acciones (o reacciones) de un comportamiento se programan reescribiendo el método action().

Cada agente tiene para sí una cola de comportamientos activos. Cuando el método anterior finaliza el programador de actividades (scheduler) lo saca de la cola o lo vuelve a colocar al final de ella.

Un comportamiento puede bloquearse (block()) hasta que lleguen más mensajes al agente; este bloqueo significa que cuando termina la acción (action()) , se debe colocar en la cola de bloqueados y solo puede salir de ella si llega un nuevo mensaje, luego de lo cual se le coloca al final de la cola de comportamientos activos.

2.4.2 Plataforma SeMoA

2.4.2.1 Historia y Características

SeMoA es una plataforma de agentes móviles desarrollada en su totalidad en JAVA, cuyo enfoque es la seguridad y basado en filosofía de código abierto. Debido a que fue desarrollada en JAVA, SeMoA funciona en todas las plataformas soportadas por el JDK 1.3 o superior.

Es un proyecto del departamento de Seguridad para Multimedia y Comunicación del Fraunhofer Institute for Computer Graphics de Alemania. Volker Roth [40] fue quien inicio el proyecto y a la fecha el equipo cuenta con alrededor de 11 miembros de los cuales podemos destacar a Ulrich Pinsdorf, Jan Peters y Peter Ebinger.

Entre los proyectos relacionados con SeMoA tenemos CODEC que es un paquete de JAVA para codificar y decodificar estructuras ASN.1 (una notación

que describe estructuras de datos para codificar, decodificar, transmitir y representar información); Earth es una extensión que permite la visualización de un agente en un contexto geográfico; Atlas es un mecanismo que permite llevar un control de donde se encuentran los agentes.

Entre las principales características de SeMoA tenemos:

- Alta portabilidad (totalmente desarrollado en JAVA)
- Verificación de integridad y autenticación de agentes
- Cajas de arena ("sandbox")
- Verificación de código y escaneo por viruses
- Bitácoras de las actividades del agente.
- Configuración de los parámetros de seguridad.
- Interoperabilidad para diferentes sistemas de agentes.

2.4.2.2 Arquitectura de seguridad de la plataforma SeMoA

La arquitectura de seguridad de SeMoA se puede comparar con un modelo de *cebolla* (de varias capas), como se puede ver en la figura 2.7 los agentes deben pasar varias capas (cuatro capas) de seguridad antes de poder ser admitidos en el sistema en ejecución y la primera clase sea cargada en el JVM.

La *primera capa* de seguridad es un protocolo de transporte como TLS y SSL(Secure Sockets Layer (SSL) y Transport Layer Security (TLS), su sucesor, son protocolos criptográficos que proporcionan comunicaciones seguras en Internet). Se utiliza la implementación que provee la infraestructura JSSE.

La *segunda capa* consiste en un serie de filtros de seguridad con canales separados para agentes entrantes y agentes salientes. Cada filtro inspecciona y procesa los agentes entrantes/salientes, rechazándolos y aceptándolos. SeMoA tiene dos pares complementarios de filtros que se encargan de las firmas digitales y el cifrado selectivo de los agentes (verificación de firmas, cifrado de

agentes entrantes, cifrado y firma de agentes salientes). Un filtro adicional al final de la serie asigna un conjunto configurable de permisos a los agentes entrantes basado en la información establecida y verificada en los filtros anteriores. Los filtros pueden ser asignados dinámicamente o durante el proceso de arranque del servidor SeMoA ya sea programado o por medio de un script.

La *tercera capa* consiste en que las clases de un agente sean cargadas por su inicializador de clases dedicado. Todas las clases cargadas son verificadas con un conjunto configurable de funciones de resumen codificado ("hash"). El resumen codificado de clases verificadas debe concordar con el correspondiente resumen firmado por el dueño del agente. Sólo clases autorizadas por el dueño del agente para ser utilizadas con su agente pueden ser cargadas en el espacio de nombres del agente.

Antes que una clase sea definida en el JVM, el bytecode de esa clase se pasa por una serie de filtros similar al de los agentes entrantes. Cada filtro puede inspeccionar, rechazar y hasta modificar el bytecode. SeMoA viene con un filtro que rechaza las clases que implementan el método finalize() (realiza acciones de limpieza antes de descartar irrevocablemente un objeto). Agentes maliciosos puede implementar este método para atacar al hilo recolector de basura en el anfitrión.

Se considera como *capa cuatro*, la creación de un área segura de ejecución para el agente que ha sido aceptado. Cada agente recibe por separado un grupo de hilos de ejecución y un inicializador de clases. El agente es deserializado por un hilo que ya está corriendo dentro del grupo de hilos del agente y se convierte en el primer hilo de ese agente. La serialización es realizada por ese mismo hilo luego que todos los hilos restantes en el grupo de hilos del agente han terminado.

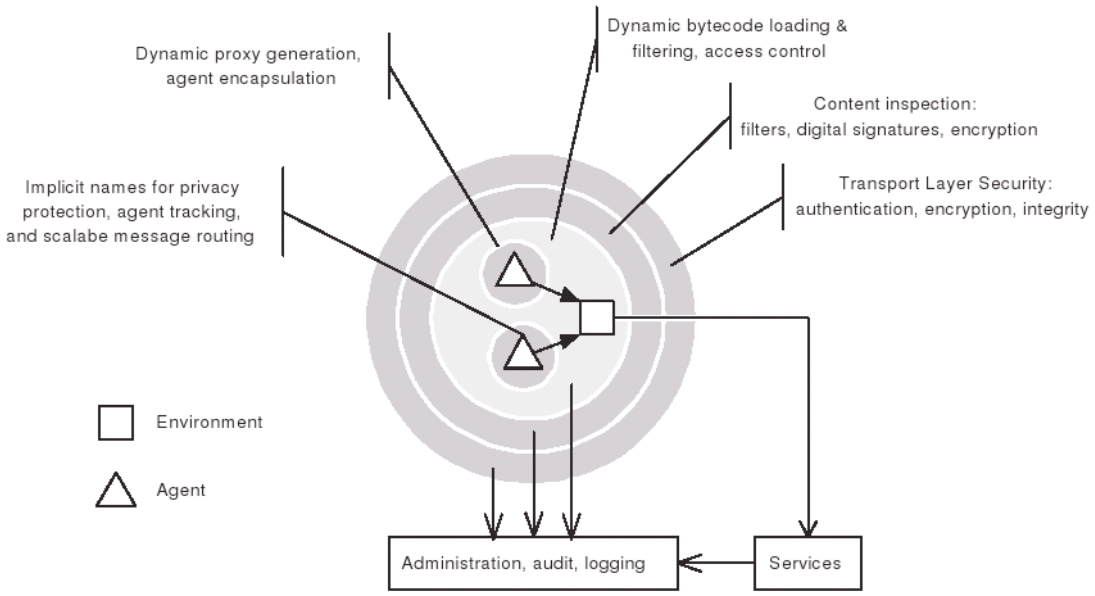


Fig. 2.7 - Diagrama de arquitectura de seguridad.
Fuente: SeMoA Details. http://www.semoa.org/about/details.html

2.4.2.2.1 Estructura del Agente SeMoA

En SeMoA, los agentes móviles son transportados como archivos JAR. La especificación JAR de Sun extiende los archivos ZIP con soporte de firmas digitales. El formato de la firma es PKCS#7[17]. Utilizando PKCS#7, SeMoA permite el uso de cifrado selectivo de los contenidos en el formato JAR para múltiples destinatarios.

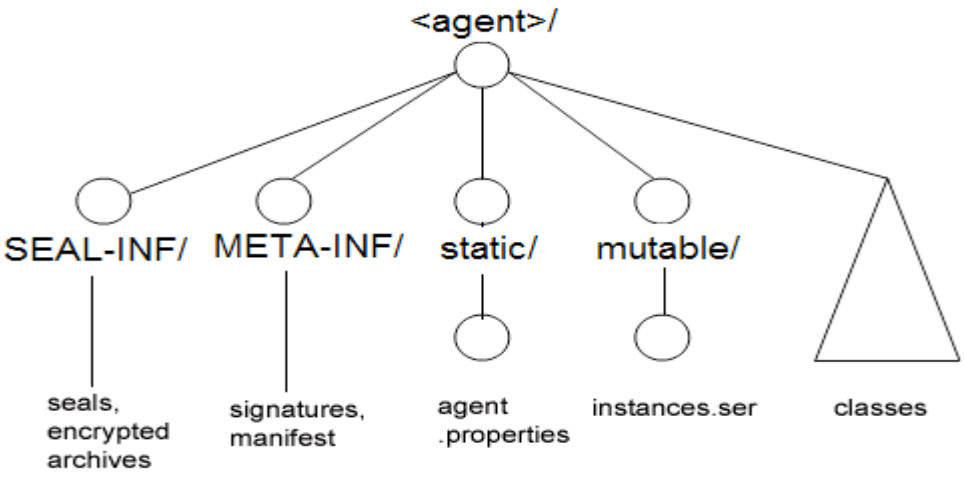
El cifrado y el descifrado son manejados de manera transparente para los agentes por los filtros de seguridad de SeMoA. Para prevenir que partes cifradas del agente sean copiadas y utilizadas en otros agentes, los filtros implementan una prueba de conocimiento no interactiva de las llaves de decifrado requeridas. O sea que el agente no participa activamente en el proceso de verificación.

Cada agente lleva 2 firmas digitales. La entidad que firma la parte estática del agente (o sea la entidad por parte de quien actúa) es tomada como la verdadera dueña de ese agente. Cada servidor que envía un agente lo firma digitalmente en su totalidad (la parte estática y la parte variable), por lo que agrega el nuevo estado del agente en su parte estática. Cada servidor es responsable de adjuntar los cambios de estado que sufrió el agente mientras se ejecuto en él.

Para hacer uso de los datos almacenados en la parte estática y variable, los agentes utilizan relaciones del tipo llaves a valores (cada llave puede tener uno o más valores).

Cuando un agente migra, su estructura es procesada por los filtros de seguridad de salida y es comprimido en un JAR para ser transportado. La versión serializada del grafo de instancias del agente (grafo de las clases heredades) también es almacenada en la estructura interna del agente, específicamente en su parte dinámica.

Los agentes pueden programar la captura de imágenes (snapshots) del estado de ejecución.



17 una sintaxis estándar para mensajes cifrados que trabaja sobre estándares como ASN.1, X.501 y X.509

Fig. 2.8 - Estructura del agente SeMoA. Fuente: SeMoA Details. http://www.semoa.org/about/details.html

La estructura del agente también contiene sus propiedades (pares de llave/valor), como el apodo legible por humanos del agente y el código fuente para cargar las clases. Las propiedades deben estar firmadas por el dueño del agente, para así ser protegidas contra alteraciones.

SeMoA, produce nombres para los agentes aplicando un resumen SHA-1 a la firma del dueño. Esto produce nombres globalmente únicos así como anónimos. Los nombres generados son utilizados en SeMoA para llevar seguimiento del agente así como para permitir el paso de mensajes independientemente de ubicación.

En el siguiente capítulo se explicará las teorías y los enfoques existentes para lograr la interoperabilidad entre plataformas de agentes móviles, haciendo especial énfasis a la interoperabilidad entre las plataformas JADE y SeMoA.

3.1 Estrategias para lograr interoperabilidad

Es conocido que para lograr interoperabilidad (en cualquier ámbito, no sólo en computación) se debe tener un conjunto bien definido de protocolos e interfaces. Para los agentes móviles se han realizado especificaciones como MASIF de OMG y FIPA, que buscaban lograr la estandarización de los ambientes de ejecución. Principalmente lo que se ha tratado de estandarizar son los puntos donde el agente entra en contacto con la plataforma [37]; algunos de ellos son:

- *La Comunicación:* Transporte de mensajes y lenguaje de comunicación. Como ejemplo: la comunicación entre agentes heterogéneos.

- *La Movilidad:* protocolos de transporte del agente, codificación del agente, traducción entre distintos lenguajes de programación. Como ejemplo la transferencia de agentes.

- *La Seguridad:* autenticación del agente. FIPA recomienda utilizar estándares ya existentes (ver 2.2.2.2).

  Lo cierto es que no existe un verdadero estándar en el punto de seguridad. Ambos estándares (MASIF y FIPA) mencionan que la seguridad es muy importante y que se debe tratar en un futuro.

- *Puntos Generales:* organización y ciclo de vida del agente e interfaces de los sistemas. Como ejemplo están la administración de los agentes, seguimiento de agentes y la estandarización de los ambientes de ejecución y plataformas de agentes.

- Basándonos en los puntos de contacto agente-plataforma podemos resumir los estándares MASIF y FIPA en la siguiente tabla 3.1:

Punto	Estándar FIPA	Estándar OMG MASIF
Administración de Agentes	Si	Si
Comunicación de Agentes	Si	No
Movilidad de Agentes	No	Si
Seguridad	No	No

Tabla 3.1 – Se muestran los puntos de contacto mencionados arriba y los estándares de agentes existentes. Fuente: David R.A. de Groot, Cross-Platform Generative Agent Migration.

De los puntos arriba detallados, hacemos especial interés en el punto de movilidad, también conocido como migración, que es sin duda lo que hace móviles a los agentes móviles y es de especial interés cuando hablamos de interoperabilidad.

A pesar de la gran variedad de implementaciones de plataformas de agentes se identifican dos nociones al momento de la migración de un agente: *la noción débil* y *la noción fuerte*.

La *noción fuerte* de movilidad es cuando el proceso del agente y su contexto de ejecución (estado de ejecución, el contador del programa) son enviados al destino. En los sistemas que soportan la migración fuerte este proceso de migración es transparente para los agentes. Es necesario explicar que para que haya migración fuerte es necesario un ambiente homogéneo, en otras palabras, que todos los nodos tengan el mismo sistema operativo, plataforma de agentes y la misma versión del lenguaje de programación en el cual están programados los agentes.

En la *noción débil* se hacen diferencia entre dos aspectos importantes del agente: *el estado del agente* y *el código del agente (*código fuente del agente y las clases de donde fue instanciado).

El estado del agente no es más que el *estado interno del agente* y los *datos privados del agente* (itinerario del agente e información recolectada después de visitar algunos nodos).

A la hora de migrar el estado completo del agente es capturado**,** *serializado* (ver 1.4.2.3) y transferido con el resto del código hacia el siguiente destino; tan pronto como llega el agente es reconstruido y ejecutado.

En plataformas como JADE y SeMoA una combinación de código y estado para la migración se implementan mediante la tecnología de serialización de JAVA; esta tecnología provee un mecanismo para transportar instancias de objetos; como estas instancias están basadas en código de programa se dice que JAVA es un lenguaje que solo soporta la movilidad o migración débil. [15]

Las especificaciones del estándar de FIPA solo han sido adoptadas totalmente por un pequeña parte de las plataformas de agentes existentes. Y ni las que lo han adoptado son interoperables sin modificaciones [37], de modo que se requiere encontrar otra(s) solución(es) para permitir la migración entre agentes de distintas plataformas.

3.1.1 Posibles Escenarios de Migración

La migración fuerte y débil no son siempre posibles. Basados en las posibilidades de la plataforma de agentes y el tipo de agentes, existen tres diferentes escenarios identificables como:

- **Escenario de Migración Homogénea**

  Es la forma más simple de migración, el origen y el destino tienen las mismas interfaces y proveen ambientes de ejecución similares para los agentes. Las interfaces son las mismas dado que el origen y el destino son parte de una misma plataforma. Para lograr la migración no es necesario cambiar el código ejecutable del agente.

  En este escenario es posible tener migración fuerte y débil. Muy pocas plataformas utilizan la migración fuerte, algunas de ellas son ARA [5], NOMADS [34], D'AGENTS [14]. Sin embargo las plataformas mayormente utilizadas están escritas en JAVA y por lo tanto utilizan migración débil; algunas de ellas son Tracy, JADE y SeMoA.

- **Escenario de Migración Cruzada (Cross-Platform Migration)**

  Para comprender este escenario es necesario tener claro el concepto de "una instancia de la plataforma de agentes", este concepto es análogo al de los lenguajes de programación, en el cual se dice que más de una instancia de una clase[18] pueden existir, en este caso múltiples instancias de una plataforma pueden existir en paralelo.

  Por ejemplo la plataforma JADE puede ser ejecutada en dos servidores diferentes. Cada una de estas instancias crea un host para el agente, de modo que la migración se da entre diferentes instancias de una misma plataforma de agentes. Es posible tener movilización cruzada entre plataformas homogéneas y heterogéneas.

[18] un objeto es una instancia de una clase

Se asume que ambas plataformas están escritas en el mismo lenguaje de programación, de modo que el código ejecutable no necesita ser cambiado, (ya que es compatible con la máquina virtual de la plataforma destino). Si la migración es entre nodos heterogéneos, es posible que las interfaces (API´s) sean diferentes lo que complica la migración.

Otra solución podría ser que la plataforma destino tenga diferentes "servidores de agentes". Cada servidor de agente soporta una interface diferente (cada una emula una plataforma diferente) y puede ejecutar los agentes que así lo requieran. Un agente puede migrar a una plataforma mientras esté presente el servidor de agentes capaz de entenderlo (ejecutarlo). Esta solución se observa en la implementación de agentes JADE sobre la plataforma SeMoA [37] y que utilizaremos más adelante para probar la interoperabilidad entre las plataformas JADE y SeMoA.

- **Escenario de Migración Heterogénea**

En este escenario los agentes se mueven entre plataformas en diferentes lenguajes de programación.

Ni la migración débil, ni la fuerte son aplicables a la serialización con JAVA, ya que el código es incompatible con el ambiente al cual él migra. El código fuente original del agente es inútil para generar código ejecutable, dado que el código compilado no es soportado por el host destino.

Este es el escenario más complicado para la migración. Una posible solución es la traducción de código, aunque debemos tener presente que no solo el código necesita traducción, sino que también los agentes

necesitan ser adaptados a las diferentas interfaces de la plataforma a la que migran.

Una solución al problema de la migración en ambientes heterogéneos es la *migración generativa (*generative migration) [15], que tiene como idea principal, el tener un centro de rediseño de agentes, donde éstos sean recreados (se crea un nuevo agente idéntico al primero, pero con las especificaciones necesarias para su migración).

Ahora que se tienen claros los tres escenarios de migración entre plataformas de agentes, es necesario decir que en esta investigación nos concentramos en la *Migración Cruzada (Cross-Platform Migration)*, ya que nuestro objetivo final es ejecutar los agentes de la plataforma JADE en la plataforma SeMoA, para lo cual es necesario seguir algunas reglas que veremos en los puntos siguientes de *Interoperabilidad en la Plataforma JADE* y *Interoperabilidad en la Plataforma SeMoA*.

3.2 Interoperabilidad de la Plataforma JADE

Como hemos dicho la forma básica de lograr la interoperabilidad es la creación y subsiguiente uso de estándares aprobados (por ejemplo los de FIPA). Generalmente estas especificaciones comprenden principalmente la infraestructura (servicios, formatos y aplicaciones de agentes), pero existen muchas diferencias que impiden lograr una total interoperabilidad entre plataformas. Muchas de estas diferencias no son nada más que ciertas elecciones de los creadores de las plataformas en aspectos no cubiertos por los estándares FIPA.

Amor, Fuentes y Pinto, afirman que *"El objetivo de FIPA es estandarizar especificaciones que permitan la comunicación entre agentes software heterogéneos. Sin embargo, FIPA no proporciona una especificación única y obligatoria, sino que ofrece varias opciones para elegir… A pesar de seguir la*

directrices dadas por FIPA, si se toman diferentes opciones y suposiciones, la interoperabilidad a través de plataformas se ve comprometida y por tanto también se reduce la reutilización de agentes software ya desarrollados en aplicaciones que impliquen distintas plataformas" [4].

Para solventar las diferencias entre plataformas que aplican los estándares de FIPA, Amor, Fuentes y Pinto proponen la implementación de una *arquitectura basada en componentes de software reutilizables* [4]. En esta arquitectura se descompone cada función del agente en componentes independientes. Funciones que forman parte del agente como sus comportamientos, sus protocolos de interacción y el manejo de mensajes (a través de un servicio de transporte de mensajes) están separados dentro de la arquitectura, de modo que es posible alterar alguno de estos componentes en tiempo de ejecución (como medida para lograr la interoperabilidad) sin que el resto se vea afectado.

3.2.1 Organización y Ciclo de vida del Agente JADE

Como hemos dicho anteriormente (ver 2.4.1.1) la plataforma de agentes JADE sigue las especificaciones de los estándares de FIPA, por lo tanto el Ciclo de Vida de un agente JADE es realmente el Ciclo de Vida de un agente FIPA. A continuación se explica como está organizado el ciclo de vida de un agente según las especificaciones de FIPA [16].

Los agentes FIPA existen físicamente en la plataforma de agentes y utilizan sus servicios para realizar sus funciones, en este contexto podemos decir que el agente es un proceso en ejecución y tienen un ciclo de vida (ciclo de ejecución) que debe ser manejado por la plataforma de agentes.

Algunas características del ciclo de vida de los agentes FIPA son:

- *Limitado a la plataforma de agentes:* un agente es físicamente manejado dentro de una plataforma de agentes y por tal motivo el ciclo de vida de un agente estático esta limitado a una plataforma de agentes especifica.
- *Independiente de la Aplicación:* el modelo de ciclo de vida es independiente del sistema y define únicamente los estados y transiciones de estados del agente en su ciclo de vida.
- *Orientado a Instancia:* en el ciclo de vida se asume que el agente es una instancia de una clase y por tal razón tiene un nombre único y es ejecutado independientemente.
- *Unicidad:* el agente solo puede estar en UN estado de su ciclo de vida y solamente en una plataforma de agentes, en un momento dado.

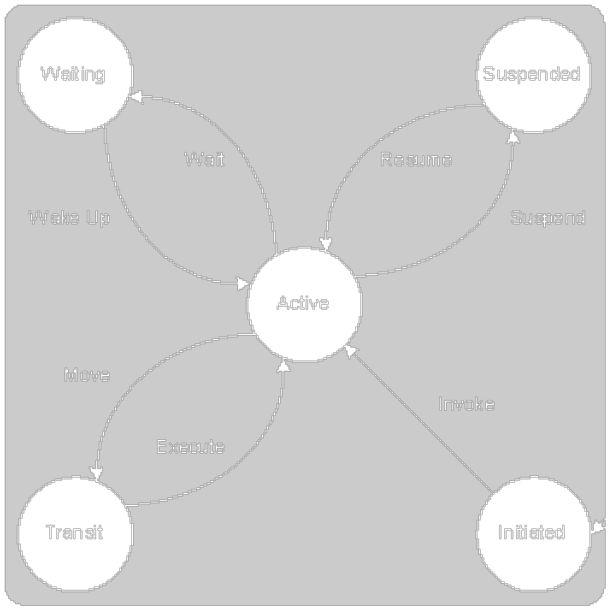


Figura 3.1 – El Ciclo de Vida de un Agente FIPA. Fuente: FIPA Agent Management Specification.

Los estados del Ciclo de Vida de los agentes FIPA pueden ser descritos como:

- *Creación (create):* se refiere a la creación o instalación de un nuevo agente.
- *Invocación (invoke):* se refiere a la invocación de un Nuevo agente.
- *Destrucción (destroy)* y *Parada (quit):* ambos se refieren a la terminación del agente solicitada por el AMS. La destrucción (destroy) no puede ser ignorada por el agente, mientras que la Parada (quit) si puede ser ignorada por éste.
- *Suspender (suspend):* coloca al agente en estado de suspendido. Este estado puede ser iniciado por el agente o por el AMS.
- *Resumir (resume):* trae al agente de estado suspendido. Este estado sólo puede ser iniciado por el AMS.
- *Espera (wait):* coloca al agente en estado de espera. Este estado sólo puede ser iniciado por el agente.
- *Despertar (wake up):* trae al agente de un estado de espera. Este estado sólo puede ser iniciado por el AMS.

Los siguientes dos estados sólo son usados por los agentes móviles.

- Movimiento (move): coloca al agente en un estado transitorio (de movimiento). Este estado solo puede ser iniciado por el agente.
- Ejecución (execute): trae al agente de un estado transitorio y lo ejecuta. Este estado solo puede ser iniciado por el AMS.

3.2.2 Ejemplo de Interoperabilidad con JADE: KODAMA y JADE

Generalmente las interacciones entre agentes se dan gracias a los protocolos de comunicación, por esto las especificaciones de KQML y las de FIPA le ponen tanto énfasis a ellos. Como JADE sigue las especificaciones de FIPA para lograr una comunicación exitosa con una agente JADE en ambas

plataformas, debe existir un Directory Facilitator (DF) y un Agent Management System (AMS), por lo tanto es imposible para una plataforma sin DF, ni AMS, interactuar con una que siga los estándares de FIPA. [23]

Para solventar las diferencias entre plataformas heterogéneas donde una de ellas aplica los estándares de FIPA, Takahashi, Zhong y Matsuno proponen la implementación de un Protocolo de Plataforma de Agentes (APP), que es un protocolo a red diseñado para suplir las demandas de la interacción en redes globales.

Takahashi, Zhong y Matsuno tienen por conclusión en su trabajo que *"La interoperabilidad con la plataforma JADE no es tan sencilla como se especifica en el estándar de FIPA y muchas veces el mismo estándar FIPA (por su rigidez en ciertos aspectos) hace que las plataformas que lo siguen NO logren la tan deseada interoperabilidad*."

3.3 Interoperabilidad en la Plataforma SeMoA

Para explicar como funciona la interoperabilidad en SeMoA, nos basaremos enteramente en el documento llamado *Mobile Agent Interoperability Patterns and Practice,* escrito por Ulrich Pinsdorf y Volker Roth, creadores de la plataforma SeMoA.

En vez de seguir el enfoque descendente de interoperabilidad (desde el estándar hacia la aplicación), los creadores de SeMoA escogieron el enfoque ascendente de interoperabilidad, basado en la *voluntad* de hacerse interoperable con otras plataformas de agentes. Para sus pruebas de interoperabilidad escogieron dos plataformas conocidas JADE y Tracy, con el fin de correr agentes de estas dos plataformas sin necesidad de modificar al agente.

Las experiencias resultantes de este proceso caen en dos categorías: en los patrones de diseño que facilita que las plataformas de agentes sean interoperables y los patrones que no la facilitan.

3.3.1 Organización y Ciclo de vida del Agente

Un agente móvil en SeMoA es un archivo JAR[19], que contiene el estado serializado del agente (y otros datos).

3.3.1.1 Migración (Salida de un agente)

El envío de un agente a otro nodo se puede resumir en los siguientes pasos:

1. El agente le pide a la plataforma que lo envíe hacia su siguiente destino.
2. SeMoA verifica que todos los hilos de ejecución (threads)[20] del agente hayan terminado de ejecutarse (la única forma de burlar esta protección es que el agente ataque el hilo recolector de basura (GC)[21] haciendo uso de una implementación maliciosa del método Finalize()[22] en los filtros de seguridad).
3. SeMoA utiliza el método Serialize de JAVA para serializar al agente y verifica que en este proceso no se hayan creado nuevos hilos (de nuevo se verifica que no haya un mal uso del método "finalize()")
4. El agente es enviado al destino deseado siempre y cuando haya pasado exitosamente todas las verificaciones de seguridad.

19 JAR o formato de Archivos Java permite empaquetar un conjunto de clases JAVA en un sólo archivo. Es usado para guardar clases JAVA compiladas (.class) y un descriptor asociada a las clases conocido como metadata.
20 Los hilos permiten dividir un programa en dos o más tareas que corren simultáneamente, por medio de la multiprogramación
21 El recolector de basura (GC) es una forma de manejo de memoria automática. Su función es colectar la memoria utilizada por objetos que no se volverán a utilizar en la ejecución del programa.
22 Este es un método llamado por el recolector de basura cuando este determina que no existen mas referencias a un objeto.

3.3.1.2 Migración (Llegada de un agente)

Antes de explicar cuales son los procesos que se deben cumplir a la llegada del agente a una plataforma SeMoA, es necesario explicar las características esenciales de todo agente que se ejecuta en SeMoA.

3.3.1.2.1 Características de los Agentes

La definición de las características del agente se encuentra en un archivo de texto, cuyas líneas son pares de la forma *Llave=Valor* y existen en él dos características obligatorias:

- La característica de *Tipo de Plataforma de Agentes*, que identifica la plataforma de agentes en la cual el agente fue creado. Como ejemplo Tracy, JADE, Aglets[23] o SeMoA.

- La característica de *Tipo de Agente*, que identifica el sistema para el cual fue creado el agente. Como ejemplo se utiliza *Java* para agentes programados en el lenguaje de programación JAVA.

3.3.1.2.2 Proceso de Deserialización

Antes de que el agente sea deseralizado y su clase constructora sea ejecutada en la máquina virtual, el archivo (JAR) debe pasar por los filtros de seguridad que tienen como función la autenticación y verificación de la integridad del agente mediante firmas digitales.

Una vez que el agente es admitido en el servidor un *cargador de clases*[24] es creado para el agente. Un hilo (thread) se encarga de invocar el método para la

[23] Aglets es una plataforma de agentes móviles escrita en JAVA, desarrollada originalmente por Mitsuro Oshima y Danny Lange en el IBM-Tokyo Research Laboratory.

deseralización del agente, luego de terminar la deserialización del agente este hilo se convierte en el primer hilo del agente.

Cuando se deserializa el agente, su nombre de clase se compara con el nombre de la clase escrito en la parte estática del agente (escrita por el dueño del agente), para prevenir la sustitución de la clase principal del agente por otra clase ejecutable (que extienda la interfase runnable de java)[25] que esté incluida en el agente.

3.3.1.2.3 Registro del Ciclo de Vida del Agente

Luego de la deserialización del agente y basado en los valores de sus características, éste es pasado al *Registro del ciclo de vida* (lifecycle registry), que no es más que una fábrica de contenedores virtuales llamados *Instancias de ciclo de vida* (lifecycle instances) que se ajustan al agente ofreciéndole un ambiente idéntico al de su plataforma nativa y que terminan encargándose de manejar su ciclo de vida.

Esta instancia (o contenedor virtual) recrea todos los componentes necesarios para que el agente crea que está corriendo en su sistema nativo y es la encargada de serializar y desearialzar al agente.

En SeMoA los ciclos de vida son definidos generalmente utilizando los siguientes métodos:

- *Start():* para comenzar su ejecución en la máquina virtual.
- *Stop():* para detener su ejecución (para migrar).
- *Suspend():* para pausar la ejecución.

[24] En el contexto de la explicación un cargador de clases es un hilo que se encarga de ejecutar la clase principal (inicial) del agente

25 La interfase runnable de JAVA se implementa en clases en las que se quiere que sus instancias sean ejecutadas en un hilo (thread).

- *Resume(ErrorCode err):* los agentes resumen su ejecución cuando falla la suspensión o en errores de migración. En los tres casos un código de error es retornado para describir el por que la ejecución fue resumida.

Luego de ajustar el ciclo de vida del agente, el hilo principal del agente hereda cuatro clases que contienen servicios nativos de SeMoA, esos servicios son:

- *Contexto de Movilidad (Mobility Context)*: provee los métodos para pedir la migración del agente u otro nodo y para saber el nombre y el resto de la información del agente.
- *Contexto de Comunicación (Communication Context):* provee los métodos para enviar y recibir mensajes.
- *Ambiente (Enviorment):* provee acceso al espacio compartido por todos los agentes.
- *Contexto de Variables (Variables Context):* provee acceso de lectura al archivo de características del agente, tomado del archivo JAR del agente.

Como todos los demás hilos del agente heredan del hilo principal, estos servicios también están disponibles para cada uno de ellos.

Por seguridad para cada agente en la plataforma SeMoA existe una *etiqueta de permiso* para prevenir que un agente acceda a servicios que están asignados a otro agente.

3.3.1.4 Adaptación de la comunicación de Agentes

La adaptación de la comunicación entre agentes tiene un problema bien marcado y es el problema de llamar a los agentes correctamente. Esto no es un problema cuando los agentes de otras plataformas se ejecutan en SeMoA, pero generalmente las plataformas de agentes utilizan esquemas basados en sintaxis URL (Uniform Resource Locator), con una sintaxis o patrón parecida

a: agente-x@lugar.com:8000/folder, donde *agente-x* es el nombre escogido por el creador del agente para el agente. SeMoA no permite escoger el nombre del agente; éste es asignado. Los nombres consisten en resumen SHA-1[26] de 20 bytes de largo.

Si un agente de otra plataforma de agentes es creado en SeMoA se calcula el SHA-1 y luego se le asigna el nombre basado en URL para que concuerde con la sintaxis de su plataforma nativa. Si al agente se le asigna el nombre *foo* en su sistema de origen, entonces un método de concordancia (mapeo) debe ser utilizado en la implementación del ciclo de vida de un agente de esa plataforma en SeMoA.

3.3.1.5 Adaptación de la Migración de Agentes

Ulrich Pinsdorf y Volver Roth establecen que la comunicación entre agentes es más sencilla que la migración, debido a que la migración está más integrada con el diseño e implementación del sistema. [37]

Las políticas de seguridad de SeMoA requieren que todos los hilos de un agente hayan sido terminados antes de permitir su migración. Mientras que en otras plataformas de agentes *esto no se llega ni a considerar*.

3.3.2 Implementación de Ciclos de Vida de Plataformas de Agentes en SeMoA

La implementación de los ciclos de vida de otras plataformas de agentes en SeMoA será directa siempre y cuando se cuente con lo siguiente: [37]

- Sea de código fuente abierto y cuente con buena documentación.
- Existencia de interfaces bien definidas para la manejar la dependencia entre los agentes y su sistema.

[26] SHA-1 o Algoritmo de Hash Seguro 1 (Secure Hash Algorithm 1) es un algoritmo criptográfico de

- Un diseño modular que anticipa implementaciones alternativas para sus módulos.
- Sea una plataforma que preste atención a la seguridad
- Sea una plataforma capaz de proveer servicios en forma de agentes y no en forma de clases que necesitan ser adaptadas o tratadas en maneras diferentes
- Modela los servicios requeridos por los agentes en forma separada y no como un solo bloque.

3.4 Interoperabilidad SeMoA-JADE

JADE se enfoca en la comunicación y cooperación entre agentes más que en la movilidad, por consiguiente la comunicación está muy desarrollada y la migración muy poca. El fin de interoperar con JADE es que los agentes pudieran correr sin necesidad de recopilación y de una manera que estos agentes JADE corriendo en SeMoA aun pudieran comunicarse con otros agentes JADE en la misma plataforma o en otra remota.

Los agentes JADE originalmente son inicializados por el *AgentToolkit* que es una interface que funciona como el punto de contacto entre el agente y la plataforma. En SeMoA se implementó esta interfase para poder mediar entre la plataforma SeMoA y el agente JADE y se hizo parte del ciclo de vida del agente JADE en SeMoA, conocido como *JADElifecycle*.

JADE soporta comportamientos. Esto permite repeticiones periódicas de una acción específica. Para poder implementar esto en SeMoA se requirió el uso de un reloj global.

La comunicación en JADE se basa en CORBA y su implementación en la plataforma JADE está separada del resto paquetes normales de la plataforma. El *JADElifecycle* de SeMoA reutiliza este paquete. Al iniciar SeMoA activa un

cifrado.

módulo que es responsable de recibir mensajes entrantes, mandar mensajes salientes y de transformar entre direcciones internas y externas.

Conclusiones sobre la Interoperabilidad SeMoA-JADE

Las pruebas realizadas por Pinsdorf y Roth confirmaron que la comunicación entre agentes JADE ejecutándose en SeMoA funciona sin problemas y mencionan como conclusión que: *"Aunque no es posible levantar alguna crítica técnica sobre el diseño de JADE, hubo problemas a la hora de la implementación de JADE en SeMoA, ya que muchas de las clases y paquetes de JADE estaban declarados con acceso protected o private sin razón obvia, lo cual se solucionó simplemente al cambiar estas clases y paquetes a acceso public".*

En el siguiente capítulo se exponen la metodología, escenarios y resultados de las pruebas realizadas para probar la interoperabilidad entre JADE y SeMoA, además de las pruebas de desempeño en JADE.

4.1 Motivación y metodología de las pruebas (Goal)

Las pruebas a exponer en este capítulo se pueden dividir en dos tipos: pruebas de interoperabilidad y pruebas de desempeño.

El objetivo de las *pruebas de interoperabilidad* es: Replicar las pruebas expuestas en el documento *Mobile Agent Interoperability Patterns and Practice,* escrito por Ulrich Pinsdorf y Volker Roth, creadores de la plataforma SeMoA, en donde se logró la interoperabilidad entre las plataformas de agentes: SeMoA y JADE (también se logró la interoperabilidad entre SeMoA y Tracy, pero no es el fin de nuestra investigación).

El objetivo de las *pruebas de desempeño* es: mostrar el comportamiento aproximado que se puede esperar en el mundo real para los agentes en ambientes donde existan muchos agentes en paralelo (pruebas de paralelismo).

4.2 Características del ambiente de prueba (Testbed)

Las pruebas tienen objetivos diferentes y por tal razón tienen características diferentes, que se definen a continuación:

- *Características de las pruebas de interoperabilidad:* no consisten en mediciones numéricas, mas bien son pruebas de concepto. Según Peter Braun [9] la interoperabilidad tiene tres aspectos a tratar: Ejecución, Comunicación y Migración; en nuestro caso para verificar estos tres aspectos se hará lo siguiente:

    - *Ejecución:* Ejecutar un agente JADE dentro de SeMoA

- *Comunicación:* Comunicar agentes usando el lenguaje ACL, en las siguientes situaciones:
  - Comunicar un agente JADE ejecutándose en SeMoA y un agente JADE en su plataforma nativa.
  - Comunicar dos agentes JADE ejecutándose en plataformas SeMoA.
- *Migración:* Migrar un agente JADE ejecutándose en SeMoA a otra plataforma SeMoA.

- *Características de las pruebas de desempeño:* consisten en la medición del tiempo de recorrido, en otras palabras el tiempo requerido para que un agente haga un recorrido circular entre los nodos de una red, saliendo de un nodo y regresando posteriormente a él. En la figura 4.1 se muestra una configuración básica para las mediciones de tiempo de recorrido, se mide el tiempo que transcurre desde la salida del agente del nodo 1 hasta su regreso a él.

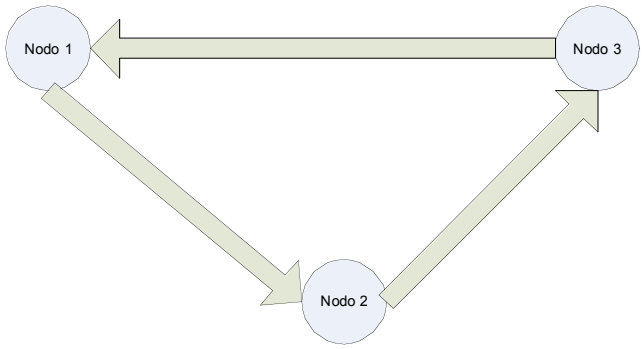


Fig 4.1 - Configuración básica para mediciones de tiempo de recorrido con tres nodos. Fuente: Los autores.

De cada medida se quiere obtener un *tiempo promedio de recorrido,* que es el tiempo total del recorrido dividido entre el número de veces en que el agente completa un recorrido circular completo.

Más adelante en este capítulo se darán los detalles de esta prueba.

4.3 Escenarios

Al existir dos tipos de pruebas podemos dividir los escenarios en dos tipos:

1. *Escenario de las Pruebas de Interoperabilidad:* para explicar los escenarios de estas pruebas se tienen que definir algunas *reglas*.

- La ejecución y migración interoperables sólo se puede dar cuando un agente de JADE es ejecutado en SeMoA y no en el caso contrario.
- Todas las comunicaciones son vía mensajes ACL.
- Las pruebas pueden ser de ámbito *local* (cuando los agentes se ejecutan una misma instancia de la plataforma y en una misma computadora), ámbito *local con múltiples instancias* (cuando los agentes se ejecutan en dos instancias de la plataforma y en una misma computadora) y ámbito *remoto* (cuando los agentes se ejecutan en dos instancias de la plataforma cada una en una computadora diferente pero conectadas vía red local).

En la tabla 4.1 se explican con mayor detalle las pruebas de interoperabilidad a realizar.

Aspecto de Interoperabilidad	**Escenario**
Ejecución	Ejecución de una agente JADE en una instancia de SeMoA.
Comunicar un agente JADE ejecutándose en SeMoA y un agente JADE en su plataforma nativa.	*Local con múltiples instancias:* un agente JADE se ejecuta en SeMoA y otro agente JADE se ejecuta en JADE, en una misma computadora.
	Remoto: un agente JADE se ejecuta en SeMoA y otro agente JADE se ejecuta en JADE, cada uno en una computadora diferente.

Comunicar dos agentes JADE ejecutándose en plataformas SeMoA.	*Local:* ambos agentes JADE se ejecutan en una misma plataforma SeMoA.
	Local con múltiples instancias: cada agente JADE se ejecuta en una plataforma SeMoA diferente
	Remoto: cada agente JADE se ejecuta en una plataforma SeMoA y en una computadora diferente
Migración	*Local con múltiples instancias:* un agente JADE se ejecuta en SeMoA y migra a una plataforma SeMoA, en una misma computadora.
	Remoto: un agente JADE se ejecuta en SeMoA y migra a una plataforma SeMoA remota.

Tabla 4.1 – Pruebas de Interoperabilidad.

2. *Escenario de las Pruebas de Desempeño:* Las medidas mencionadas en el punto 4.2 de este capítulo se repiten variando las configuraciones, como se muestra en la tabla 4.2.

Escenario	**Tipo de Nodo**	
Pruebas de Paralelismo		
Medición de tiempo de recorrido de un número determinado de agentes entre al menos dos nodos JADE.	Homogéneos: Mismo Sistema Operativo (Linux-Linux, Windows-Windows) y versión de JVM	Heterogéneos: Diferentes Sistemas Operativos (Windows-Linux) y misma versión de JVM

Tabla 4.2. - **Pruebas de Desempeño.**

4.4 Detalles de las pruebas de Interoperabilidad

Como hemos explicado anteriormente las pruebas de interoperabilidad sólo se dan en un sentido, que significa que un agente JADE se ejecute en SeMoA, ya que a la fecha es imposible hacer lo contrario debido a no existe ningún modulo de compatibilidad con agentes SeMoA para JADE.

En Junio 26 del 2006 salió al publico una nueva distribución de SeMoA (que al momento de escribir este capítulo es la más reciente) en la cual los creadores incluyeron el paquete de Interoperabilidad de SeMoA-JADE, de modo que ahorran al usuario el trabajo de configuración de la plataforma para que sea interoperable con JADE, lo único que necesita saber el usuario es como cargar las variables iniciales de la plataforma en el archivo de configuración a utilizar, esto se explica en el ANEXO B – Parte I.

4.5 Detalles de las Pruebas de Desempeño

Para entender las pruebas de desempeño es necesario detallar el proceso de medición de tiempo en JAVA, para luego entrar a los detalles específicos de la configuración de software que se utilizó.

4.5.1 Medición del Tiempo en JAVA

Para medir intervalos de tiempo usamos el método de JAVA System.currentTimeMillis(), el cual devuelve el numero de milisegundos que han pasado desde el 1ero de Enero de 1970. Diferentes implementaciones de la JVM proveen diferentes niveles de precisión.

Una de las sugerencias de los creadores de JAVA en la medición de tiempo utilizando System.currentTimeMillis() es que es posible encontrar un error en el orden de 10ms aproximadamente dependiendo del Sistema Operativo.

4.5.2 Detalles de las plataformas

Existen algunos detalles relacionados a la plataforma JADE que pensamos es necesario describir.

- *Detalles de JADE:* la plataforma JADE tiene migración intra-plataforma por defecto, de modo que la migración de los agentes JADE

sólo ocurre entre un Contenedor Principal (plataforma JADE) y un Contenedor Secundario (plataforma JADE añadida a la principal); Con el fin de que la migración de agentes JADE fuera más parecida a la de SeMoA (migración entre plataformas de agentes o Main-Containers), decidimos utilizar el paquete llamado *Inter-Platform Mobility Service 1.1.1* (la versión original utilizada fue 1.0, pero esta fue modificada varias veces gracias a nuestro reporte de errores al autor, más información vea ANEXO B Parte II) creado por Joan Ametller-Esquerra y Jordi Cucurull-Juan estudiantes de doctorado de la Universidad Autónoma de Barcelona, que permite la migración entre contenedores principales vía HTTP. Este paquete está para su descarga gratuita en la página de JADE.

Detalles del Agente JADE de Medición de Desempeño: el agente de medición fue proporcionado por Jordi Cucurull-Juan, quien lo desarrolló para realizar las pruebas de su investigación de *Movilidad y Seguridad de Agentes de Software* [22] presentada como tesis de maestría en Mayo de 2006. Este agente mide el tiempo (tiempo total, tiempo promedio por agente, tiempo total por iteración y el tiempo total por iteración y por agente) que toma mover un número X de agentes a un numero N de nodos, Y veces. Para más detalles sobre este agente ver ANEXO B Parte II. En la figura 4.2 se puede ver una salida típica del agente de medición de desempeño.

```
Starting the test...

################## Deleting file .\t8df8081.jar
All agents returned! Test finalized.

Total time: 1922 msec
Average time per agent: 1922 msec
Total time per iteration: 1922 msec
Average time per iteration and agent: 1922 msec
```

Fig 4.2 – La Salida de una ejecución del agente de desempeño con parámetros de una instancia y una iteración.

4.6 Configuración de Hardware y Software

La siguiente tabla 4.3 provee información que muestra las características de las computadoras utilizadas en nuestras pruebas.

Hardware		
	Pc #1	**Pc #2**
Modelo	Dell Optiplex GX400	Dell Optiplex GX400
Procesador	Intel Pentium 4 1700Mghz	Intel Pentium 4 1700Mghz
Chipset	Intel i850	Intel i850
Total de Memoria	256 MB (400Mhz)	256 MB (400Mhz)
Fast Ethernet Adapter	3Com 3C920 Integrated Fast Ethernet Controller	3Com 3C920 Integrated Fast Ethernet Controller
Ethernet Switch	8Port Fast Ethernet Switch	
Software		
Sistemas Operativos	Windows XP Professional 2002 Service Pack 2 / Ubuntu Linux 6.06 LTS	Windows XP Professional 2002 Service Pack 2 y Ubuntu Linux 6.06 LTS
JVM	1.5.0_07 / 1.5.0_06	1.5.0_07 / 1.5.0_06
JADE	3.4	3.4
SeMoA	060626	060626
Nombre	UsiJavier	UsiJorge

Tabla 4.3 – Características del Equipo Utilizado en las Pruebas.

4.7 Resultados de las Pruebas

En esta parte se exponen los resultados de las pruebas, debemos recodar que las pruebas de Interoperabilidad son *pruebas de concepto* con resultados no

son cuantificables, por otro lado las pruebas de desempeño son numéricas y por lo tanto es necesario explicar sus resultados mediante gráficas y análisis de los datos.

4.7.1 Resultados de las pruebas de Interoperabilidad

4.7.1.1 Ejecución

La ejecución de un agente JADE dentro de SeMoA es transparente al usuario, como se ve en la figura 4.3 es una ventana de JAVA normal. Es de notarse que aunque es un agente de JADE el nombre del agente es una versión cifrada como cualquier agente SeMoA.

El único requisito es que en el archivo de configuración de SeMoA se debe activar el servicio de compatibilidad con la opción JADE_COMPAT. Ver ANEXO B Parte I.

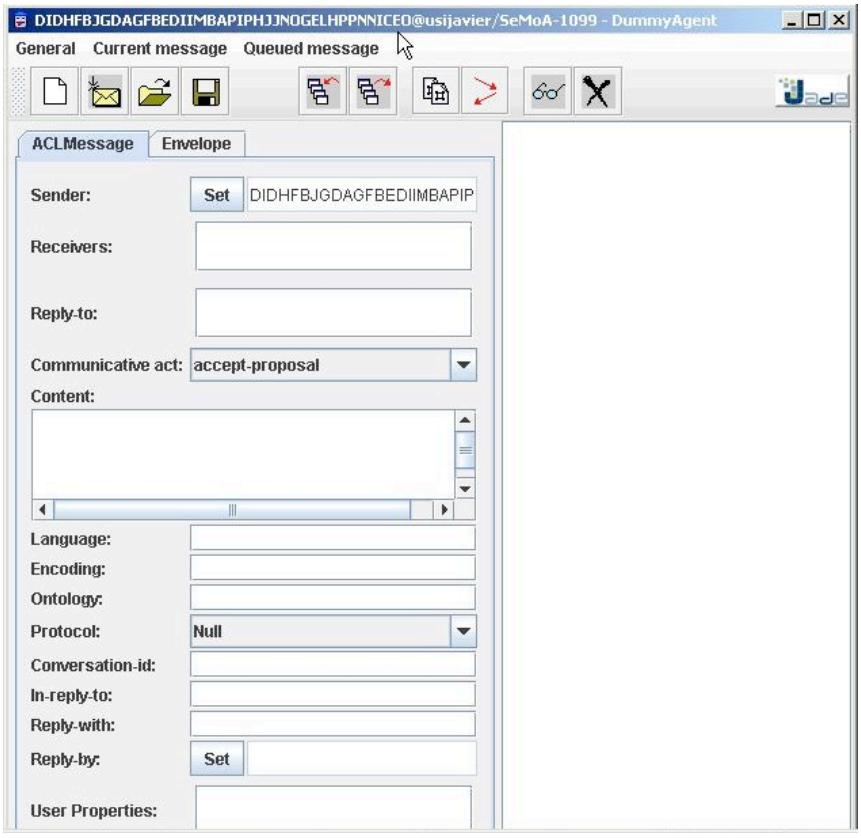


Fig. 4.3 – Dummy Agent de JADE ejecutándose en SeMoA.

4.7.1.2 Comunicación

Para estas pruebas se utilizo el agente "Dummy", que viene incluido con la plataforma JADE y que se usa para enviar y recibir mensajes ACL. En la figura 4.4 se muestra a un agente Dummy ejecutándose en una plataforma JADE, mientras recibe un mensaje enviado por un agente Dummy ejecuntadose en SeMoA.

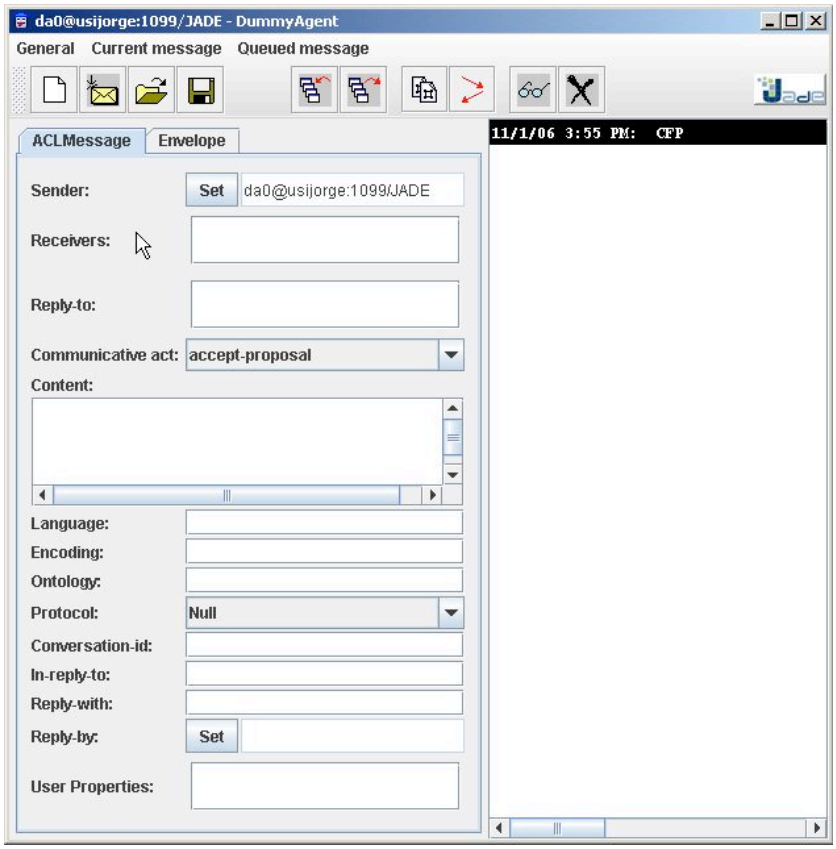


Fig 4.4 - Un Dummy Agent en una plataforma JADE recibiendo un mensaje de un Dummy Agent ejecutándose en SeMoA.

En la Tabla 4.4 se resumen todos los resultados de las pruebas de comunicación.

Creador del Mensaje	**Receptor del Mensaje**	**Detalles**	**Resultado**
Agente JADE en **SeMoA**	Agente JADE en **SeMoA**	*Local*	Funciona
Agente JADE en **SeMoA**	Agente JADE en **SeMoA**	*Local con múltiples instancias*	No Funciona. (ResponseMessage is not OK). Ver Anexo C para guía de cómo realizar esta prueba.
Agente JADE en **SeMoA**	Agente JADE en **SeMoA**	*Remoto*	No Funciona. (ResponseMessage is not OK)
Agente JADE en **SeMoA**	Agente JADE en **JADE**	*Local con múltiples instancias*	Funciona
Agente JADE en **SeMoA**	Agente JADE en **JADE**	*Remoto*	Funciona.
Agente JADE en **JADE**	Agente JADE en **SeMoA**	*Local con múltiples instancias*	No Funciona. (ResponseMessage is not OK)
Agente JADE en **JADE**	Agente JADE en **SeMoA**	*Remoto*	No Funciona. (ResponseMessage is not OK)

Tabla 4.4 - Resultados de las Pruebas de Comunicación.

4.7.1.2.1 Análisis de los Resultados

ResponseMessage is not OK

Para lograr entender este error, primero debemos entender como es la comunicación correcta entre dos agentes Dummy y así comparar. En la figura 1 del Anexo D se observa una comunicación entre dos agentes Dummy ejecutados sobre JADE. El contenido en rojo es el mensaje enviado y el azul es la respuesta de la plataforma receptora. Esta conversación fue exitosa. Hay que notar como se envió el mensaje y luego se obtuvo una respuesta usando el protocolo HTTP (sin ACL) por parte de la plataforma receptora.

A diferencia, en la figura 2 del Anexo D observamos como el mensaje fue enviado correctamente, mientras que la respuesta es vacía.

La plataforma SeMoA contesta con un mensaje vacío como se observa en la figura 3 del Anexo D. Esto es lo que causa el WARNING en la plataforma que envió el mensaje como se observa en la figura 4.5.

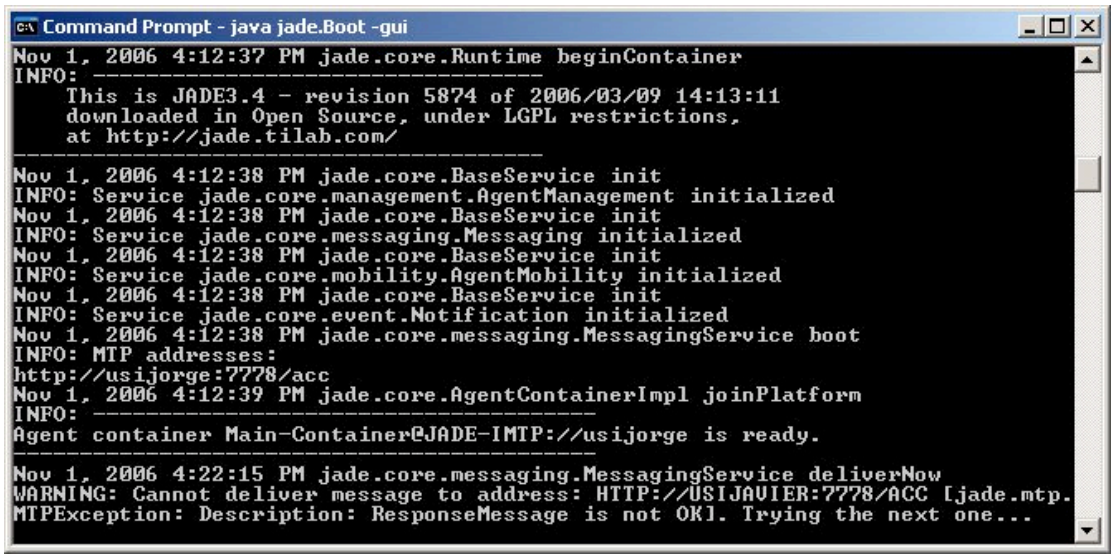


Fig 4.5 – Error de Response Messages is Not OK, en la plataforma JADE en una consola DOS.

Dirección incorrecta de remitente.

Al realizar las pruebas de comunicación nos percatamos que cuando el agente Dummy ejecutado en SeMoA enviaba el mensaje era recibido sin problemas por el agente Dummy ejecutado en JADE, como se ve en la figura 4.6, el problema resulto al tratar de responder, ya que no era posible contestarle directamente porque la dirección del remitente estaba incorrecta.

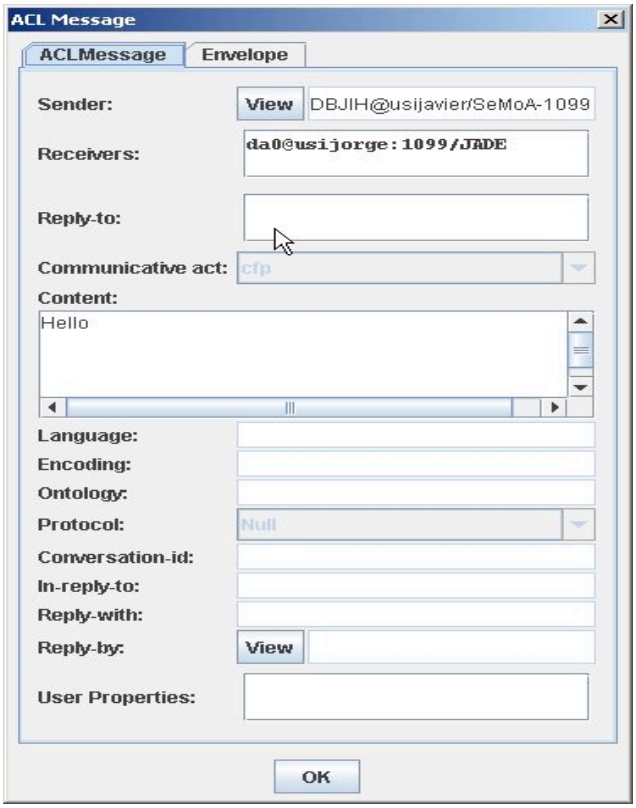


Fig 4.6 – El mensaje enviado por el Dummy Agent Ejecutándose en SeMoA.

En la figura 4.7 y la figura 4 del Anexo D vemos la información incorrecta en el remitente *[LJAVA.LANG.STRING@...]"*, este es un error dentro de la implementación de JADE en la plataforma SeMoA, causado por una mala impresión de un objeto que contiene un arreglo de cadenas. La figura 5 del Anexo D muestra como debe ser enviada la dirección.

Fig 4.7 – Error en la dirección del remitente.

Este error es posible notarlo de otras maneras, una de ellas es utilizando el Sniffer Agent (agente que *escucha* todas las conversaciones que se dan en una plataforma) como se puede ver en la figura 4.8, la otra manera de verlo es en la consola DOS en la que se levanta el JADE, esto se puede ver en la figura 4.9.

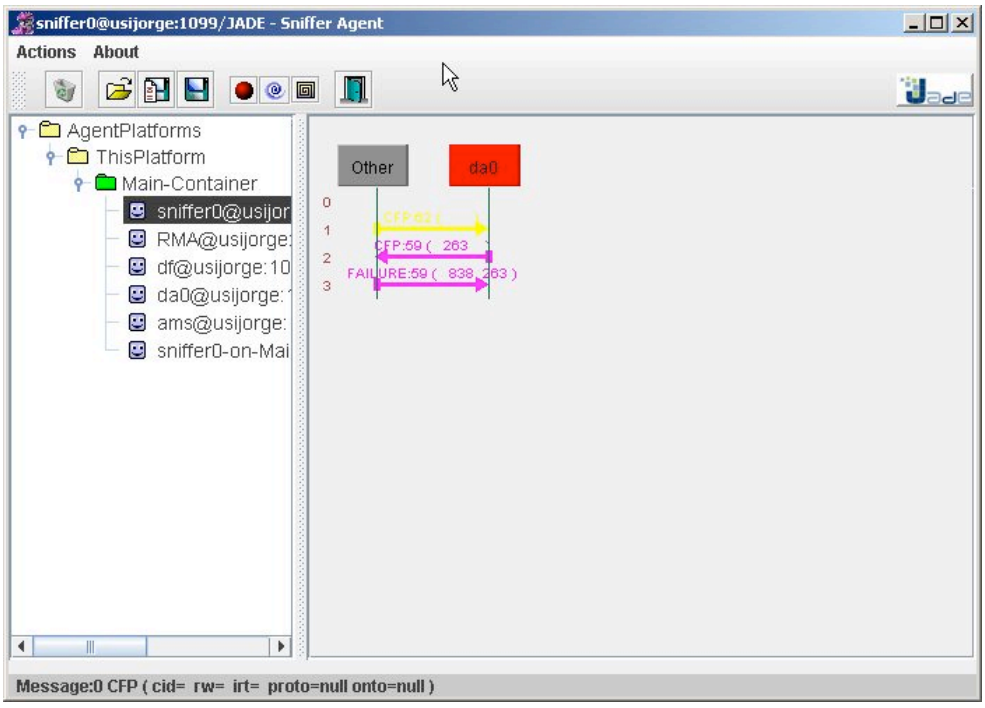


Fig 4.8 – Error al dar respuesta al mensaje, visto desde el Sniffer Agent.

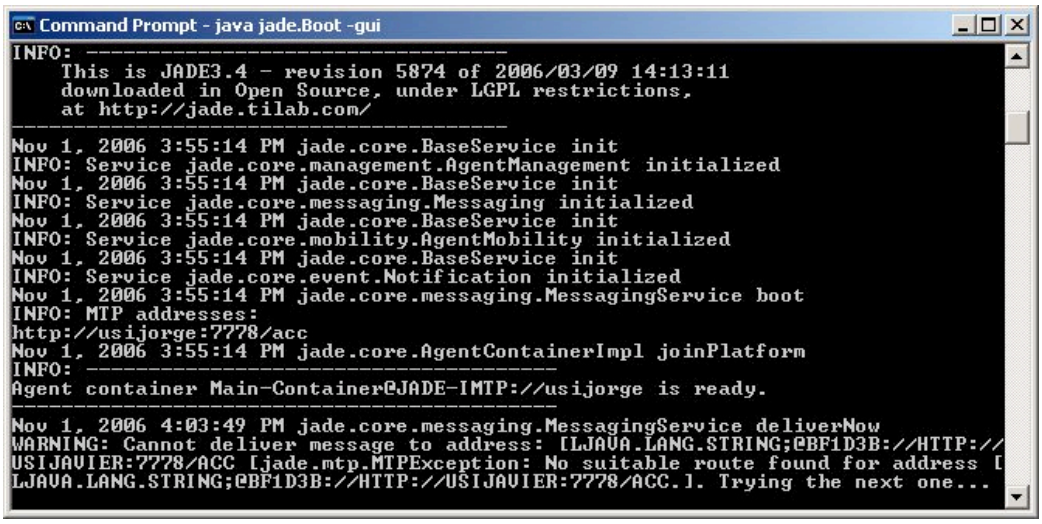


Fig 4.9 – Error al dar respuesta al mensaje, visto desde la consola de JADE.

Luego de estudiar todas las situaciones anteriores y los resultados, llegamos a la conclusión de que la plataforma SeMoA es incapaz de recibir mensajes enviados por medio del servicio HTTP. Además en que envía su remitente incorrectamente.

Para solucionar estos problemas habría que realizar un estudio profundo de la implementación del manejador de mensajes de HTTP de la plataforma SeMoA.

4.7.1.3 Migración

La migración de un agente JADE ejecutándose en SeMoA a otra plataforma SeMoA es un punto que no está escrito en el documento de Pinsdorf y Roth y que fue comentado a nosotros vía email (Ver ANEXO A) y al momento de este escrito aún estamos en espera de una respuesta de parte del grupo de desarrollo de SeMoA con la explicación.

Cabe destacarse que la única migración propiamente explicada en el documento de Pinsdorf y Roth es la de un agente Tracy ejecutándose en

SeMoA hacia otra plataforma SeMoA (Tracy no fue considerada en esta investigación pero la proponemos para futuras investigaciones).

4.7.2 Resultados de las pruebas de Desempeño

El objetivo de las pruebas de desempeño en JADE es tener la medida del paralelismo, en otras palabras obtener una medida de cómo se comportarían un numero determinado de agentes migrando entre nodos de red.

En la realización de estas pruebas replicamos una de las pruebas descritas en la tesis de maestría de Jordi Cucurrul Juan [22], en la cual se hicieron pruebas con 1, 10 y 100 agentes, con recorridos de 1, 10, 100 y 1000 iteraciones.

Para obtener los valores para cada combinación hicimos cinco (5) veces cada prueba y sacamos un valor promedio, este valor es el que se puede ver en las Tablas 4.5, 4.6 y 4.7 y graficado en las Gráficas 4.1, 4.2 y 4.3.

En la figura 4.8 presentamos los resultados expuestos en la tesis de Jordi Cucurrul, estos valores varían entre 48 y 921 milisegundos.

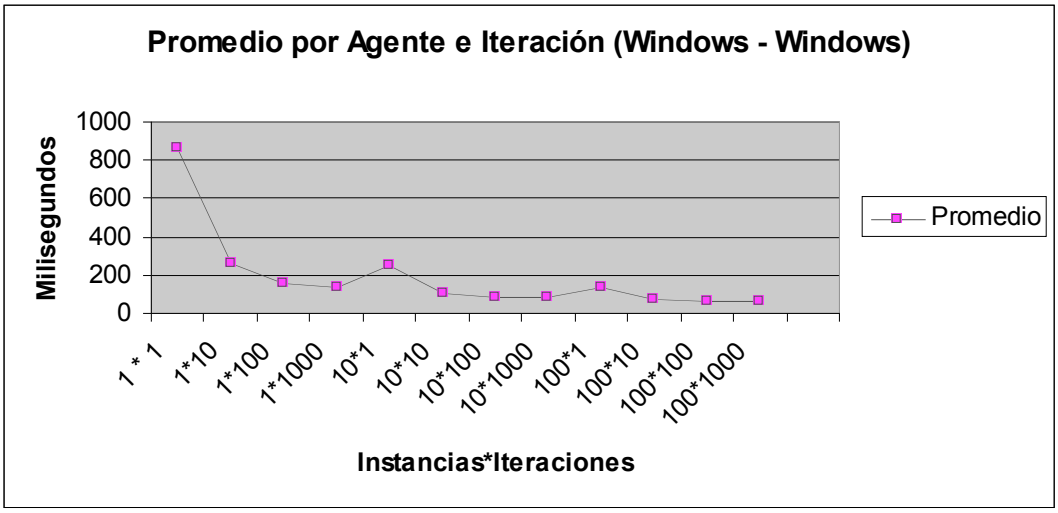
Instàncies	Iteracions	Temps total	Temps per agent i iteració
1	1	921 ms	921 ms
1	10	2557 ms	256 ms
1	100	14063 ms	141 ms
1	1000	116178 ms	116 ms
10	1	1988 ms	199 ms
10	10	8624 ms	86 ms
10	100	70798 ms	71 ms
10	1000	622461 ms	62 ms
100	1	10358 ms	104 ms
100	10	56237 ms	56 ms
100	100	482145 ms	48 ms
100	1000	4774170 ms	48 ms

Fig 4.10 – Resultados del uso del Agente de Desempeño. Fuente: Jordi Cucurull Juan. Contribucio a la Mobilitat I Seguretat Dels Agents Software. Tabla 7.1, pagina 81.

A continuación los valores promedios resultados de nuestras pruebas para Windows – Windows (tabla 4.5) y la grafica de estos valores (Grafica 4.1)

Pruebas Windows a Windows			
Instancias	**Iteraciones**	**Tiempo Total Promedio**	**Tiempo por instancias e iteraciones**
1	1	861.40	861.4
1	10	2559.60	255.96
1	100	15266	152.66
1	1000	134615.80	134.62
10	1	2451.80	245.18
10	10	10545	105.45
10	100	85445	85.44
10	1000	828583.20	82.86
100	1	13301	133.01
100	10	70419.20	70.42
100	100	662240	66.22
100	1000	6628373	66.28

Tabla 4.5 – Resultados para Windows – Windows

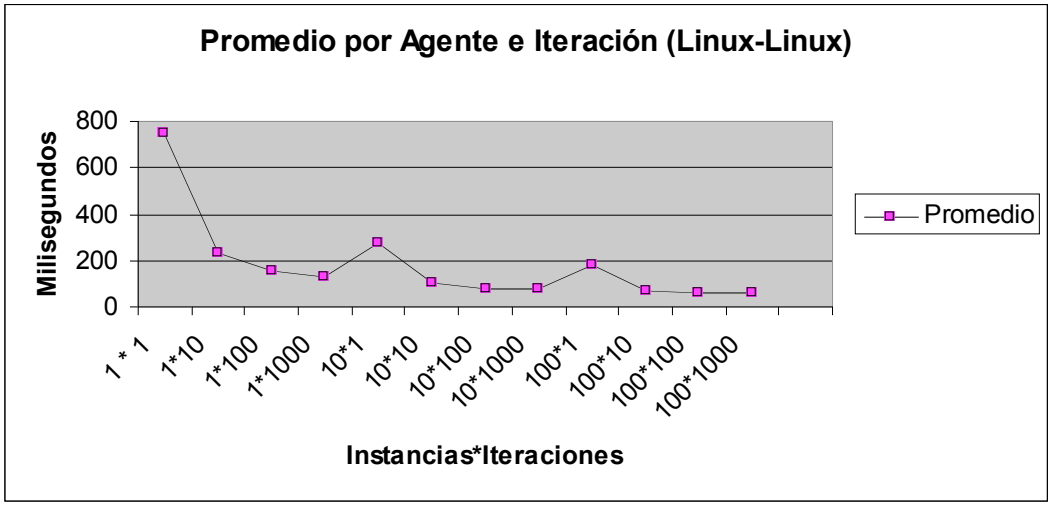


Grafica 4.1

A continuación los valores promedios resultados de nuestras pruebas para Linux – Linux (tabla 4.6) y la grafica de estos valores (Grafica 4.2)

Pruebas Linux a Linux			
Instancias	**Iteraciones**	**Tiempo Total**	**Tiempo por instancias e iteraciones**
1	1	751.20	751.20
1	10	2364.4	236.44
1	100	15178.6	151.79
1	1000	132616	132.62
10	1	2743.8	274.38
10	10	10021.8	100.22
10	100	78862	78.86
10	1000	762617.6	76.26
100	1	17855.2	178.55
100	10	71721.8	71.72
100	100	617026	61.70
100	1000	5649849.8	56.50

Tabla 4.6 – Resultados para Linux - Linux

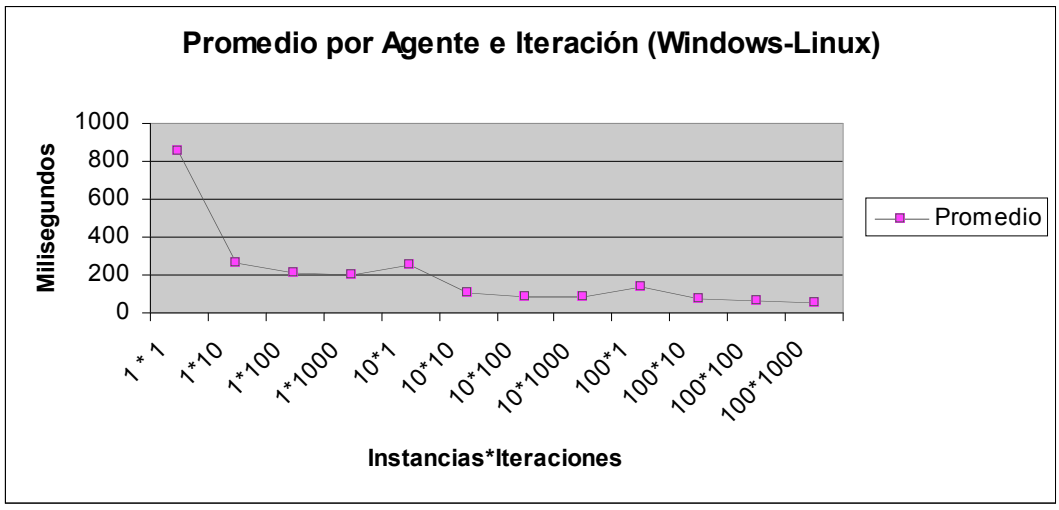


Grafica 4.2

A continuación los valores promedios resultados de nuestras pruebas para Windows – Linux (tabla 4.7) y la grafica de estos valores (Grafica 4.3)

Pruebas Windows a Linux			
Instancias	**Iteraciones**	**Tiempo Total Promedio**	**Tiempo por instancias e iteraciones**
1	1	851.40	851.40
1	10	2585.80	258.58
1	100	21258.60	212.59
1	1000	204658.40	204.66
10	1	2559.80	255.98
10	10	10483.20	104.83
10	100	83388	83.39
10	1000	810575.60	81.06
100	1	14092.40	140.92
100	10	68897.20	68.90
100	100	634752.80	63.48
100	1000	5760298.80	57.60

Tabla 4.7 – Resultados para Windows - Linux



Grafica 4.3

4.7.2.1 Conclusiones de las pruebas de desempeño

- De nuestros resultados podemos decir que los mejores tiempos los tuvimos en ambiente Linux-Linux, luego Windows-Linux y por último Windows-Windows.

- En comparación a los resultados obtenidos por Jordi Cucurrul-Juan podemos decir que en nuestros resultados se pueden observar el mismo comportamiento:

  El número de agentes es directamente proporcional al tiempo total de ejecución e inversamente proporcional al tiempo promedio por iteración y por agente.

- Como era de esperarse los tiempos de ejecución son afectados por la carga del computador, es de notar que su influencia es grande, o sea que el desempeño se degrada apreciablemente (en muchos casos los tiempos se duplicaban). En un ambiente de producción podemos afirmar que un sistema de agentes requerirá una cantidad de recursos (CPU, Memoria, Red, etc).

- La diferencias existentes entre los resultados obtenidos por Jordi Cucurrul-Juan y los nuestros radican en que el utilizó PC's con una capacidad de procesamiento mayor (más rápidas), las usadas por el eran de 2.0 GHz y las nuestras de 1.7 GHz.

Nota: En adición a los resultados numéricos que obtuvimos de las pruebas es necesario destacar que gracias a las pruebas que realizamos pudimos colaborar con el descubrimiento de errores en el Inter-Platform Mobility Service (IPMS), estos descubrimientos están documentados en el ANEXO B Parte II.

Conclusiones

- La eficiencia de una plataforma de agentes en términos de interoperabilidad esta íntimamente ligada a su diseño, puesto que este debe ser flexible o modular previendo futuras modificaciones y permitiendo la reutilización de código. En esta investigación hemos visto que la plataforma SeMoA acepta agentes de JADE, gracias a su diseño modular.

- La estandarización y creación de normas es un proceso necesario en toda tecnología, en el caso de los agentes, los estándares que han existido no llegan a cumplir con todos los detalles para que dos plataformas de agentes sean totalmente interoperables. Esperamos que el nuevo estándar que vera la luz pública en Diciembre de 2006 sea la solución que tanto espera la tecnología de agentes.

- Es necesario que exista buena documentación, clara y concisa, para ver implementaciones reales de las plataformas, como es el caso de JADE que es utilizada grandemente gracias a su disposición de documentación, una lista de usuarios activa y su facilidad para programar, por su parte SeMoA, que es una plataforma bien diseñada y estructurada, por falta de documentación, no esta tan diseminada como debería.

- Los resultados experimentales obtenidos en los puntos de pruebas de interoperabilidad y pruebas de desempeño nos han permitido documentar detalladamente y replicar trabajos investigativos previos.

- Se debe tomar en cuenta que el tiempo dispuesto para la realización de este trabajo investigativo se agotó dejando algunas pruebas sin realizarse, principalmente por falta de documentación y de apoyo de los

autores de la plataforma SeMoA, razón por la cual las hemos dejado como trabajos futuros.

Trabajos Futuros

Existen detalles que no se pudieron completar en este trabajo, que pensamos es necesario explicar para dar pie a futuros investigadores, entre ellos:

- *Migración de un agente JADE ejecutándose en SeMoA a una plataforma SeMoA:* para lograr esto es necesario tener una explicación detallada de cómo hacerlo por parte de los creadores de la plataforma.
- *Ampliar las pruebas desempeño para incluir medidas con agentes movimiento diferentes tipos de carga*: En otras palabras obtener una medida de cómo se comportaría un agente móvil llevando diferentes tipos de carga (payload), para esto ya teníamos preparado un agente de cálculo de tiempo ver ANEXO E.
- *Probar la interoperabilidad de SeMoA y Tracy*: este punto al igual que el primer punto de migración depende en gran medida de que los creadores de la plataforma SeMoA, pero seria interesante probar la migración de agentes Tracy en SeMoA.

Otros temas que pensamos podrían ampliarse en futuras investigaciones:

- Agentes inteligentes con estructura BDI y con razonamiento deductivo con JBOSS o implementaciones de Prolog.
- Agentes móviles que sean capaces interactuar con base de datos, haciendo consultas o transfiriendo el resultado de peticiones (querys).
- Utilización de Agentes móviles para aprovechamiento de la Web semántica o de servicios Web.
- Utilización de Agentes inteligentes para simulación de sistemas.

Recomendaciones

A la Universidad Tecnológica de Panamá:

- Incentivar al estudiante a desarrollar una investigación como trabajo de graduación desde los primeros años de la carrera.
- Revisar los planes de estudio con miras a insertar materias como diseño de experimentos, que se concentren en el análisis cuantitativo y cualitativo de datos (principalmente resultados de experimentos).
- Adecuar las materias del plan de estudio a la realidad de la carrera que se estudia, dando ejemplos concretos con los cuales el estudiante pueda ver la necesidad de comprender la esencia de las materias que estudia.

Referencias Bibliográficas

[1] Abdalla, M.; Cirne, W.; Franklin, L. *Security Issues in Agent Based Computing*. 1997.

[2] Agentlink III. *Agent Technology: Computer as Interaction. A Roadmap for Agent based Computing*. 2005. ISBN 085432-845-9

[3] Agentlink III. *Agent Technology Roadmap: Overview and Consultation Report*. 2004.

[4] Amor, M.; Fuentes, L; Pinto, M. *Interoperabilidad entre Plataformas de Agentes FIPA: Una Aproximación Basada en Componentes*. Universidad de Malaga.

[5] ARA (Agents for Remote Action). http://wwwagss.informatik.uni-kl.de/Projekte/Ara/index_e.html

[6] Bieberstein, N. et al. *Service-Oriented Architecture Compass*, Pearson 2006.

[7] Botía J. *La Plataforma de Agentes JADE (Java Agents Development Environment)*. Escuela de Primavera de Agentes, Patrocinado por Agentcities.es, Imaginática 2005, Universidad de Sevilla. 2005

[8] Braun, P. *Mobile Agents Work Group – Charter*. IEEE FIPA Standards Committee. 2005.

[9] Braun, P.; Trinh, D.; Kowalczyk, R. *Improving the Migration Performance of Jade: Integration of Kalong into Jade*. Swinburne University of Technology. Faculty of Information and Communication Technologies

[10] Braun, P.; Rossak, W. *Mobile Agents*. Morgan Kauffman Publications. 2004. ISBN: 1-55860-817-6.

[11] Brustoloni, J. *Autonomous Agents: Characterization and Requirements*, Carnegie Mellon Technical Report (CMU-CS-91-204). 1991. Pittsburgh: Carnegie Mellon University.

[12] Caire, G. *JADE Tutorial: JADE Programming for Beginners*. TILAB. 2003.

[13] Corchado, J. *Modelos y Arquitecturas de Agentes*. Universidad de Salamanca.

[14] D'AGENTS. http://agent.cs.dartmouth.edu/

[15] De Groot, David R.A. *Cross-Platform Generative Agent Migration*.Vrije Universiteit Amsterdam. 2004.
http://www.iids.org/publications/DRAdeGrootMScThesis.pdf

[16] FIPA. FIPA Agent Management Specification.
http://www.fipa.org/specs/fipa00023/XC00023H.html

[17] Hefeeda, M.; Bhargava, B.. *On Mobile Code Security*. 2001.

[18] Huhns, M.N, Singh, M.P. *Agents and Multi-agent Systems: Themes, Approaches, and Challenges*. Morgan Kaufmann Publishers. 1998,

[19] IEEE-FIPA. http://www.fipa.org/about/fipa_and_ieee.html

[20] IEEE-FIPA Subgroups. http://www.fipa.org/subgroups/

[21] Janicke, H.;, Siewe, F.; Jones, K.; Cau, A.; Zedan, H.. *Analysis and Run-time Verification of Dynamic Security Policies*. 2005.

[22] Jordi Cucurull Juan. *Contribucio a la Mobilitat I Seguretat Dels Agents Software*. Universitat Autonoma de Barcelona. Departament d'Enginyeria de la Informacio i de les Comunicacions. Mayo 2006

[23] Ken'ichi Takahashi, Guoqiang Zhong, Daisuke Matsuno, *Interoperability between KODAMA and JADE using Agent Platform Protocol*.Kyushu University.
http://www.agentcities.org/Challenge02/Proc/Papers/ch02_16_takahashi.pdf

[24] Labrou, Y., Fininand, T., Peng, Y. *The Interoperability Problem: Bringing together Mobile Agents and Agent Communication Languages*. Proceedings of the Hawaii International Conference On System Sciences. 9199, Maui, Hawaii.

[25] Lange, D. B.; Oshima, M. *Seven Good Reasons for Mobile Agents*. Communications of the ACM, Vol. 42, No. 3.1999.

[26] Levina, T. *Mobile Agents (Term Paper on Expert Systems)*. Queen's University, Canada. 2001.
http://www.glennshafer.com/courses/downloads/levina.pdf

[27] Mestras Pavon, J. *Estandarización de Sistemas de Agentes*. Dep. de Sistemas informáticos y Programación. Universidad Complutense de Madrid. 2001.

[28] Milojicic, D; Laforge, W; Chauhan, D. *Mobile Object sand Agents (MOA)* The Open Group Research Institute.

[29] Misikangas, P.; Raatikainen, K. *Agent Migration between Incompatible Agent Platforms*. Department of Computer Science, University of Helsinki http://www.cs.helsinki.fi/research/monads/papers/icdcs2000/index.html

[30] Muller, I; Braun, P; Rossak, W. *Integrating Mobile Agent Technology into an e-Marketplace Solution*. Friedrich-Schiller-University Jena

[31] NIST. *Mobile Agent Security*. NIST Special Publication 800-19. October 1999.

[32] NIST. National Vulnerability Database (NVD). http://nvd.nist.gov/statistics.cfm.

[33] NIST. *Underlying Technical Models for Information Technology Security*. NIST Special Publication 800-33 . 2001.

[34] NOMADS. http://citeseer.csail.mit.edu/suri00overview.html

[35] Pérez, Jesús Arturo. *Agentes Móviles: Programación, Seguridad y Diseño*. Universidad de Oviedo. 2000.

[36] Pinsdorf, U. *A Formal Approach for Interoperability between Mobile Agent Systems and Component Based Architectures*. Fraunhofer Institute for Computer Graphics.

[37] Pinsdorf, U.; Roth, V. *Mobile Agent Interoperability Patterns and Practice*. Fraunhofer Institute for Computer Graphics. http://www.volkerroth.com/download/Roth2002c.pdf

[38] Samper, J. Agentes Inteligentes. http://www.wikilearning.com/agentes_inteligentes-wkc-5095.htm

[39] Schrool, Andre Peter. *An Agent Architecture for Mobile Network*

Services: Design and Implementation. University of Victoria. 1997

[40] SeMoA. *Secure Mobile Agents - Motivation*. http://www.volkerroth.com/proj-semoa.html

[41] SeMoA. *SeMoA Details*. http://www.semoa.org/about/details.html

[42] Sparta Isso Opensource Projects. *Self-Protecting Mobile Agents (SPMA)*. http://www.isso.sparta.com/research/projects/

[43] Stan, F.; Art, G. Is it an Agent, or just a Program?. http://www.msci.memphis.edu/~franklin/AgentProg.html

[44] Vigna, G. *Mobile Agents: Ten Reasons For Failure*. Reliable Software Group. Department of Computer Science. University of California, Santa Bárbara. 2004.

[45] Vigna, G.; Orso, A.; Harrold, MJ. Mobile Agents *Security through Static/Dynamic Analysis (MASSA)*. 2001.

[46] Wooldridge, M.; Jennings, N. R. *Intelligent Agents - Theories, Architectures, and Languages*. LNAI-Verlag. 1995. ISBN 3-540-58855-8 890.

ANEXOS

ANEXO A

Información sobre Interoperabilidad en SeMoA

Email De Ulrich Pinsdorf Donde envía el add-on para utilizar agentes JADE en SeMoA y donde nos dice que es posible migrar agentes JADE de una plataforma SeMoA a otra.

From : Ulrich Pinsdorf <ulrich.pinsdorf@igd.fraunhofer.de>
Sent : Thursday, February 9, 2006 10:50 AM
To : Javier Sanchez-Galan <j_sgalan@hotmail.com>
Subject : Re: Using Semoa and Jade Agents

tesis | Inbox

Attachment : SeMoA-JADE-3.2.zip (0.04 MB)

Dear Javier,

please excuse this heavy delay of my answer. I'm very sorry for that!

In this mail I send you a JAR file which allows interoperability with JADE. This package will be a feature in the next SeMoA distribution. My colleague Jan Peters is actually compiling all new developments. The new distribution will be published in the next weeks.

I'm very interested in the results of your investigation. Have you published papers on this topic? What is the goal of your experiments? It would be very interesting to have a dialog by email on this topic.

Just a few words on the JAR file:
- Please extract the contents into your SeMoA installation directory.
- It would be wise to compare the files in your semoa/etc directory with the according files in the JAR archive and take just the additions for interoperability. Otherwise you'd reconfigure your running SeMoA platform.
- The JADE classes are patched versions of the original JADE 3.2. The only changes I made is to switch some method classifiers from 'protected' to 'public'. This was done due to classloading restrictions. I thought this would be a legal modification, no other changes, esp. no internal JADE programming had been altered.
- You need a JADE 3.2 in your classpath. Moreover we use IIOP via HTTP. Maybe you have to download additional stuff from the JADE website for that.
- Make sure, that the SeMoA classes are at the beginning of your classpath (or more precisely proceeding the original JADE classes). This is necessary for applying the patched classes, they need to be found first.
- Start the SeMoA platform as usual.
- Issue 'source etc/rc.jade' on the command line. I'd suggest to add this command as last line in file etc/rc. This would invoke the interop features automatically.
- Check out the file etc/rc.jade. Here you'll find a number of aliases for starting jade agents. Type 'jade-dummy' to start the well-known JADE DummyAgent.
- If you want to start your own agent, please make an alias for this agent similar to the examples in etc/rc.jade and run it from command line using your alias.

Two JADE agents running on different SeMoA platforms are able to communicate. It is also possible to communicate between a JADE agent running on SeMoA and a JADE agent running on a native JADE platform. JADE agents running in SeMoA benefit from all SeMoA security features. They are as secure as native SeMoA agents.

JADE agents running in SeMoA are able to migrate to other SeMoA platforms! This is a bit tricky. Please ask me if you need this feature.

Have fun and - please - report me your experiences. Feel free to ask me when you encounter any problem.

Best regards,
Ulrich Pinsdorf

ANEXO B

Parte I: JADE y SeMoA

Archivo de Configuración de SeMoA Explicado

En SeMoA existen por defecto 4 archivos de configuración, demo1, demo2, demo3 y demo4. Cada uno de ellos diferentes configuraciones de puertos. Hay que destacar que para que se puedan comunicar dos plataformas SeMoA es necesario que estas tengan archivos de configuración diferentes. Estos archivos se encuentran en el directorio /etc en el directorio donde esta instalado el SeMoA, tiene nombres como demo1.conf, demo2.conf etc.

```
#
# Original Version
#
# Author  : Volker Roth
# Revision: $Revision: 1.11 $
# Date    : $Date: 2006-04-13 16:55:34 +0200 (Thu, 13 Apr 2006) $
#

#Archivo de Llaves de certificados digitales
setenv KEYSTORE       file:///${semoa.etc}/demo2.jks

#Archivo de configuraciones de servicios de SeMoA
setenv SHIP_FILE      file:///${semoa.etc}/ship2.conf

#Ruta a la Imagen Providence.jpg, utilizada si se usa el TouristAgent (Agente Turista)
setenv POSTCARD_FILE  ${semoa.base}/lib/postcard/providence.jpg

#Puertos de los servicios de SeMoA
setenv SHIP_PORT     55002
setenv RAW_PORT      55012
setenv RAWS_PORT     55022
setenv RECEIPT_PORT  55032
setenv POD_PORT      55042
setenv PODS_PORT     55052
setenv RSHD_PORT     55062
setenv SSHD_PORT     55072
setenv HTTPD_PORT    8082
setenv HTTPSD_PORT   8882

#Para Iniciar el Servicio de Comptabilidad con JADE
setenv JADE_COMPAT   enable
```

Un Archivo de Configuración (demo2.conf), la última línea activa el modulo de compatibilidad con JADE

Parte II: Agente de Desempeño – Tiempo Vs. Numero de Instancias e Iteraciones - JADE

Agente de Desempeño en JADE

Para poder el agente de desempeño tiene que haber al menos dos participantes donde uno será el iniciador y el otro el receptor.

El Iniciador es el Generador de agentes o PC en donde comienzan las iteraciones. El receptor es donde terminan las iteraciones.

Un ejemplo de cómo usar el IPMS es:

En una consola DOS en el Iniciador

C:\>java jade.Boot -jade_core_migration_IPMS_migration_timeout 15000 -jade_core_migration_IPMS_migration_timeout_responder 30000 -services jade.core.mobility.AgentMobilityService;jade.core.migration.InterPlatformMobilityService aPerfomance:PerformanceAgent("10" "1" “C:\Platforms\jade\classes\my.properties" "MAPerformance")

aPerfomance = nombre del agente
PerformanceAgent = clase que calcula el performance
10 = numero de iteraciones
1 = numero de agentes (instancias de la clase del agente)
C:\Platforms\jade\classes\my.properties = ruta al archivo donde se encuentra la ruta de los nodos.

Nota: Es necesario explicar que JADE solo resuelve nodos por nombre (no funciona por dirección IP), por lo que se recomienda que los nombres de las PC’s que se estén usando para la prueba hayan sido escritos de antemano en el archivo de hosts.

Este archivo se encuentra en *Windows\System32\drivers\etc\hosts* en Windows y en */etc/hosts* en Linux.

MAPerformance = clase del agente

En una consola DOS en el Receptor

C:\>java jade.Boot -jade_core_migration_IPMS_migration_timeout 15000 -jade_core_migration_IPMS_migration_timeout_responder 30000 -services jade.core.mobility.AgentMobilityService;jade.core.migration.InterPlatformMobilityService

Detalles del Agente de Desempeño

El agente de desempeño tiene como salida tiempos en milisegundos, la forma de calcularlos es:

Calculo el tiempo total:
Tiempo Total = Tiempo de llegada del agente – Tiempo de salida del agente.

Calculo del tiempo promedio por agente:
Tiempo Promedio = Tiempo Total / Numero de Agentes

Calculo del tiempo promedio por iteración:
Tiempo Promedio = Tiempo Total / (Numero de iteraciones)

Calculo del tiempo promedio por agente y por iteración:
Tiempo Promedio = Tiempo Total / (Numero de iteraciones)(Numero de Agentes)*

Descubrimiento de errores en el Inter-Platform Mobility Service (IPMS)

26 de Septiembre de 2006: Gracias a nuestras pruebas en Linux sale una versión 2 del Codelocator.java, en el que se elimina el "Class Clash", por error en el sistema de cifrado.

From : Jordi Cucurull Juan <jcucurull@deic.uab.cat>
Sent : Monday, September 25, 2006 5:49 AM
To : Javier Sanchez Galan <j_sgalan@hotmail.com>
Subject : Re: Sobre Benchmark

Hola Javier,

te quería hacer una pregunta. Resulta que estaba mirando el código del IPMS i probé los tests de performance. El caso es que me he encontrado que con agentes concurrentes me quedaban JARs sin borrar en el directorio donde ejecutaba los tests.

Entonces he repasado el CodeLocator.java y he descubierto un error bastante grande que era el causante de ello. El problema es que yo hice estos tests hace unos meses y no me di cuenta de ello. Vosotros que los habeis usado, os habeis encontrado con esto que te digo de que os quedasen ficheros JAR sin borrar? (concretamente tantos JAR sin borrar como agentes). Gracias.

From : Jordi Cucurull Juan <jcucurull@deic.uab.cat>
Sent : Tuesday, September 26, 2006 2:58 AM
To : Javier Sanchez Galan <j_sgalan@hotmail.com>
Subject : Re: Benchmark y otras cosas

Hola Javier,

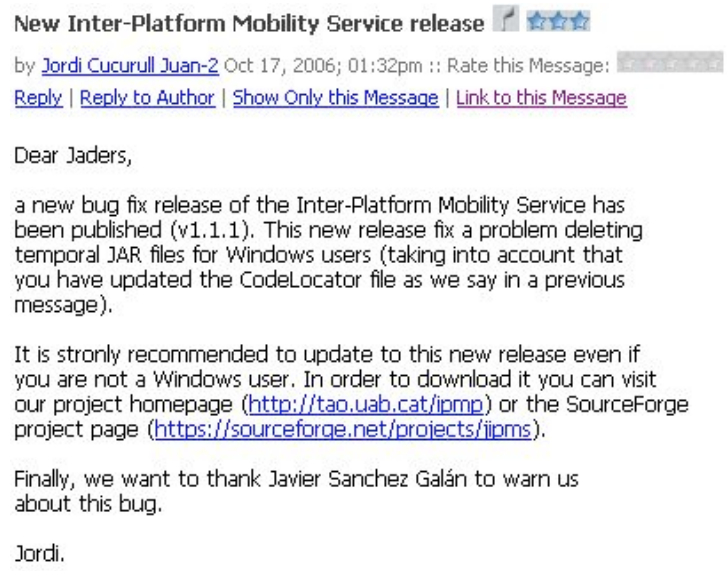
voy a ir comentando/contestando tus respuestas:

> 5- Nosotros tambien notamos que quedaban JAR sin borrar, me quede
> pensando pues varias pruebas con 100 instancias (repitiendo las que
> ustedes habian hecho), tenia casi 2500 jars en el directorio $Home, de
> hecho pense que esto era un efecto del problema de Timeout que nos
> salio al tener 100 instancias.
>
> Ahora mismo no recuerdo bien el error, pero tan pronto llegue a al
> lab, lo repito y si tengo error te mando una captura de pantalla, a
> ver si sabes que puede estar pasando
>
> Cuando puedas sube el nuevo codigo fuente para probarlo

Esto del error de los JAR me sorprendió mucho, porque yo estuve corrigiendo problemas de este tipo durante bastante tiempo. Lo que pasa es que sólo ocurre cuando tienes más de un agente con las mismas clases (porque tiene que ver con un sistema de hashes que usamos para no tener un montón de jars iguales). Lo más raro es que no lo viese al hacer los tests de performance estos que te pasé.

Ya he subido un CodeLocator.java corregido en el Subversion de SourceForge. Concretamente lo puedes ir a buscar a /trunk/src/jade/core/management/CodeLocator.java. Debes tener los fuentes de la plataforma Jade y sustituir este fichero en ellos y recompilarla. Entonces debería ir bien!

17 de Octubre de 2006: Gracias a nuestras pruebas en Windows sale una versión 1.1.1 del IPMS



New Inter-Platform Mobility Service release

by Jordi Cucurull Juan-2 Oct 17, 2006; 01:32pm :: Rate this Message:

Reply | Reply to Author | Show Only this Message | Link to this Message

Dear Jaders,

a new bug fix release of the Inter-Platform Mobility Service has been published (v1.1.1). This new release fix a problem deleting temporal JAR files for Windows users (taking into account that you have updated the CodeLocator file as we say in a previous message).

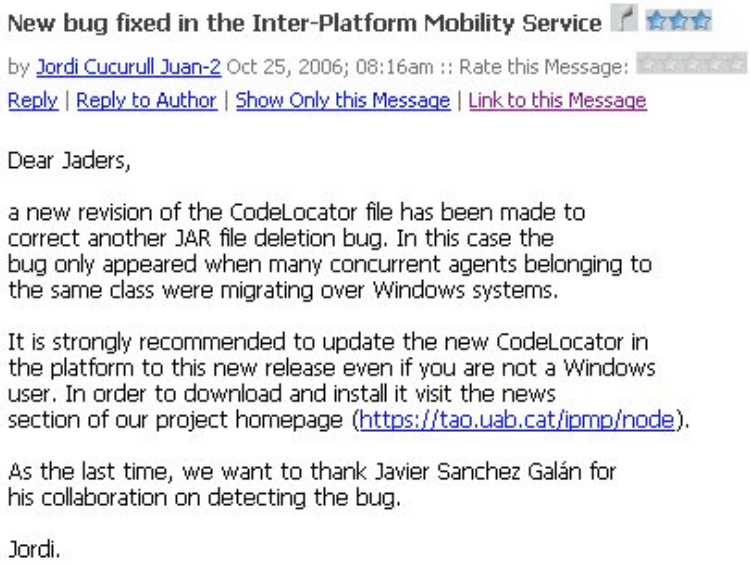
It is stronly recommended to update to this new release even if you are not a Windows user. In order to download it you can visit our project homepage (http://tao.uab.cat/ipmp) or the SourceForge project page (https://sourceforge.net/projects/ipms).

Finally, we want to thank Javier Sanchez Galán to warn us about this bug.

Jordi.

Captura del mensaje mandado por Jordi Cucurrul-Juan a la lista de usuarios de la plataforma JADE agradeciendo el encontrar el error de borrado de archivos en Windows.

25 de Octubre de 2006: Gracias a nuestras pruebas en Windows sale una versión 3 del Codelocator.java



New bug fixed in the Inter-Platform Mobility Service

by Jordi Cucurull Juan-2 Oct 25, 2006; 08:16am :: Rate this Message:

Reply | Reply to Author | Show Only this Message | Link to this Message

Dear Jaders,

a new revision of the CodeLocator file has been made to correct another JAR file deletion bug. In this case the bug only appeared when many concurrent agents belonging to the same class were migrating over Windows systems.

It is strongly recommended to update the new CodeLocator in the platform to this new release even if you are not a Windows user. In order to download and install it visit the news section of our project homepage (https://tao.uab.cat/ipmp/node).

As the last time, we want to thank Javier Sanchez Galán for his collaboration on detecting the bug.

Jordi.

Este es una captura del mensaje mandado por Jordi Cucurrul-Juan a la lista de usuarios de la plataforma JADE agradeciendo el encontrar el error de borrado de archivos.

ANEXO C
Prueba local y múltiples instancias entre dos agentes JADE en SeMoA

Para lograr realizar esta prueba se tiene que tener en cuenta cuatro puntos:

1. Un agente JADE es identificado por el nombre que se le asigna y por el nombre de la plataforma en la cual se encuentra. Por ejemplo, "usi@hostname:1099/JADE", en donde "usi" es el nombre del agente y "hostname:1099/JADE" es el nombre de la plataforma (el número de puerto es parte del identificador de la plataforma).

2. La plataforma SeMoA al ejecutar un agente JADE abre los puertos utilizados por la plataforma JADE (1099, 7778). Además utiliza como nombre predeterminado para la plataforma "hostname/SeMoA-1099". Hay que notar que en ese nombre el "1099" no está indicando el puerto a utilizar. [27]

3. La primera instancia ya está utilizando los puertos 1099 y 7778, por lo que la segunda instancia utiliza puertos aleatorios que permiten diferenciarlas.

4. Las dos instancias reconocen el nombre de "hostname/SeMoA-1099" como el suyo.

En base a lo anterior encontramos el siguiente problema: ¿Cómo indicar a qué plataforma queremos enviar el mensaje?

---

[27] **URL** significa ***U**niform **R**esource **L**ocator*, es decir, localizador uniforme de recurso. Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos por su localización. El formato general de un URL es:

protocolo://usuario:contraseña@máquina:puerto/directorio/archivo

Para resolver este problema, luego de levantar la segunda plataforma hay que verificar que puertos está utilizando. El nombre seguirá siendo el mismo entre ambas plataformas, pero el mensaje se envía utilizando el puerto aleatorio.

Sino se toman en cuenta las observaciones anteriores, entonces el mensaje siempre será eliminado por la plataforma por no encontrar el agente a quien se le envía el mensaje.