Universidad de Los Andes
Facultad de Ingeniería
Postgrado en Computación

Propuesta de un sistema de gestión de de servicios multiagente para el SCDIA

Elaborado por: Ing. Víctor Bravo
Tutor: Dr. José Aguilar

Mérida – 2003

INTRODUCCIÓN

Los sistemas multiagentes (SMAs) son sistemas de software que agrupan un conjunto de cualidades pertenecientes a los tópicos más avanzados que en el área de computación se tratan en la actualidad. Estos tópicos tienen que ver con el uso de algoritmos inteligentes, sistemas distribuidos, programación lógica, ontologías para conversaciones, hilos de ejecución, máquinas virtuales, metalenguajes, interoperabilidad e integración de sistemas, entre otros.

Esta conjunción de conceptos y tecnologías, hace posible dar respuestas a problemas complejos, a través de la construcción de sistemas autorregulados y dinámicos, que deben integrarse con aplicaciones legadas[1], plataformas y sistemas heterogéneos.

La adaptación de SMAs dentro de plataformas tecnológicas ya operativas, no puede considerarse como un problema trivial. La heterogeneidad trae como consecuencia un extenso y difícil trabajo de acoplamiento de interfaces; aunado a esto, el nuevo paradigma basado en agentes trae consigo nuevos conceptos que deben ajustarse a los ya existentes. Por ejemplo, cuando se habla de "conversaciones" entre agentes, es necesario trasladar este concepto al contexto de computación distribuida en el cuál se manipulan objetos, mensajes, protocolos, procesos, con la finalidad de implantar la comunicación entre dos entes que llamaremos agentes. La necesidad de contar con un conjunto de servicios que brinden conexión con recursos y aplicaciones, que ofrezcan las cualidades de los sistemas distribuidos, tales como interoperabilidad, migración, transparencia, seguridad, entre otras, y que a su vez permitan la gestión de un conjunto de agentes, puede constituirse en una herramienta indispensable para la integración del paradigma de agentes a las plataformas computacionales de las organizaciones de hoy en día.

En este sentido, este trabajo se propone el desarrollo de un marco operativo para SMAs con las características antes mencionadas. Así, se desarrolla

[1] Las aplicaciones legadas son aplicaciones operativas. Cuentan con una utilidad real, por lo tanto es inconveniente su reemplazo o desecho.

e implementa un sistema de gestión que permita la comunicación entre recursos, aplicaciones, y agentes. Este trabajo forma parte del Proyecto Agenda Petróleo[18]. Este proyecto está integrado por profesores y estudiantes de la Universidad de los Andes, Mérida. Tiene como principal objetivo el desarrollo de un Sistema de Control Distribuido Inteligente basado en Agentes (SCDIA). Particularmente, este trabajo tiene como objetivo proponer el nivel de gestión de servicios del SCDIA.

El Sistema de Gestión de Servicios (SGS) propuesto en este trabajo, está estructurado en tres niveles, que corresponden en sentido de abajo-arriba con los conceptos de componente, sistema distribuido y agente respectivamente. Cada uno de estos niveles brinda servicios y posee cualidades asociadas a estos conceptos. En la capa más alta se encuentran los agentes de gestión de servicios, que administran, controlan y supervisan todas las acciones del SMA. El modelado de estos agentes se hizo haciendo uso de la metodología MAS-CommonKads. Esta metodología permite modelar los agentes, sus tareas, las comunicaciones entre agentes, entre otras cosas. Dentro del marco del proyecto se extendió la metodología agregando dos nuevos modelos, uno que describe la coordinación entre los agentes y otro que describe el componente inteligente. Además se extendieron los modelos de tareas y de agentes para que fuese posible la inclusión de tareas inteligentes[14]. La metodología permite describir las características generales de los agentes, que luego serán implantadas en el proceso de construcción del SMA. Para los agentes, se escribió un conjunto de ontologías relacionada con los conceptos, predicados, secuencias y acciones que los agentes manejan entre sí. El aspecto ontológico permitió la construcción de conversaciones basados en significados, que es una de las características interesantes de los SMAs.

La capa intermedia brinda servicios de migración, nombramiento, seguridad, interoperabilidad y transparencia, características inherentes a los sistemas distribuidos. Para ello se utilizó y extendió la plataforma JADE (Java Agent Development Enviroment)[3], una plataforma multiagente escrita totalmente en Java que brinda posibilidades de integración vía CORBA y RMI.

Finalmente, la capa inferior propone un esquema basado en componentes, que integra las dos tecnologías actualmente más conocidas en esta área, como lo son COM (Component Object Model)[22] y JavaBeans[16,20]. Para esta capa también se utilizó la tecnología JNI (Java Native Interface)[20], que permitió una conexión eficiente con los recursos y aplicaciones de cada nodo en el sistema. La unión de estas tres capas conforma el SGS propuesto en este trabajo.

Este trabajo está organizado como sigue: en el primer capitulo se revisan los aspectos teóricos más importantes que sirvieron para el diseño e implementación del SGS. En el capitulo dos, se presenta el SCDIA, un modelo de referencia para construcción de sistemas multiagentes para plataformas de automatización y control industrial para el cuál se diseño originalmente el SGS. En el capítulo tres, se expone el diseño de cada una de las capas del SGS. Siguiendo la metodología MAS-Common-Kads extendida[13,14] se describen los diferentes modelos que servirán como herramienta para la construcción del sistema. El capítulo cuatro muestra el proceso de implementación, desarrollo de los modelos, uso de herramientas y resolución de problemas. El quinto capítulo, se dedica a presentar un caso de estudio, que ilustra los servicios, funcionamiento y eficiencia del SGS dentro del marco del modelo. En la última parte, se presentan las conclusiones, donde se resumen las características más importantes del trabajo realizado, y se plantean las recomendaciones.

CAPITULO I: Nociones sobre Sistemas Multiagentes, Sistemas distribuidos y Componentes

La teoría de agentes se muestra como el paradigma de la computación de los próximos años[4,23]. La idea de contar con un conjunto de características agrupadas coherentemente dentro de una entidad, permite lograr la resolución de problemas complejos dentro de organizaciones con plataformas tecnológicas heterogéneas y complejas. Los aspectos más importantes de este paradigma se describen en este capitulo. También se revisan algunos de los aspectos teóricos de los sistemas distribuidos, y de la teoría de componentes, conceptos que también fueron tratados en el desarrollo de este trabajo.

1. Sistemas Multiagentes (SMAs)

Cuando se habla de agentes en computación, no existe un consenso sobre su definición, no se tiene una descripción general aplicable a todos los casos. El concepto de "agente" tiene asociado un número variable de cualidades, que, aunque cada una de ellas tiene una definición clara, no tienen ni una única interpretación, ni una sola forma de implementarse.

Una de las características más importantes que se le atribuye a los agentes es la autonomía, la capacidad de tomar decisiones en función de un estado interno[23]. Los agentes por su naturaleza son autónomos, cualidad que puede establecerse según la perspectiva que se tenga del problema. Si el agente percibe el estado del ambiente, y analiza su estado interno, eventualmente podría tomar una acción sin tener que existir una orden expresa, en este caso puede decirse que el agente es autónomo desde cierta perspectiva. También podríamos decir que no es autónomo, ya que en este caso el agente está recibiendo una orden indirecta determinista, solapada por el ambiente, por ejemplo, una configuración del estado, que se sabe a priori, disparará una acción de un determinado agente. Por otra parte, la manera en que esa capacidad de percepción y toma de decisiones son llevadas a cabo, también es un aspecto importante a considerar. La percepción puede ser vista como la lectura de

variables de la memoria local, o en cambio, puede ser vista como procesos complejos de mediciones de variables, que involucran diferentes sistemas, componentes y hardware, como es el caso, de las plataformas para control de procesos.

La interpretación e implementación de las características de los agentes puede significar la diferencia en el logro de de las metas propuestas. Sin embargo, la teoría de agentes, ofrece desde todo punto de vista, una visión clara para la construcción de sistemas computacionales. La integración de diversas disciplinas y áreas de investigación que convergen en un concepto coherente, brinda posibilidades reales de solución a problemas complejos.

1.1. Características de los Agentes

Podemos resumir las características con la que debe contar un SMA bajo los siguientes aspectos:

1.1.1. Autonomía:

Según varios autores la autonomía es la principal característica de los agentes. Weiss en su libro “Multiagent Systems”, dice:”autonomía es la noción central de agencia” [23], esto lo argumenta diciendo que los agentes son autónomos en la medida en que actúan sin la intervención humana ni de otros sistemas externos. Se puede dar una noción general de autonomía, definiéndola como la capacidad que tiene un agente de tener un comportamiento propio, y reaccionar a los estímulos externos basándose en su estado interno. Cuando se va a la práctica, ésta característica puede ser modelada mediante el uso de hilos de ejecución (threads) para cada agente, atribuyéndole de esta manera un comportamiento independiente, lo cual marca una diferencia notoria con los objetos. Cada agente recibe y percibe señales del ambiente o de otros agentes, las analiza utilizando sus mecanismos internos, que pueden ser desde sencillas sentencias sí/entonces hasta complejos sistemas expertos dinámicos que utilizan reglas difusas.

1.1.2. Comunicación:

La teoría de agentes se hace más robusta y útil, cuando se habla de sociedades de agentes. Un solo agente puede hacer pocas cosas, pero una sociedad de agentes puede llegar a resolver problemas complejos. La comunicación entre agentes tiene entre sus objetivos acercarse a lo que es la comunicación entre personas, a través de intervenciones dinámicas con oraciones y frases que contienen significados.

La capacidad de cada agente de conversar utilizando un lenguaje basado en ontologías y realizar intervenciones asíncronas, constituye un paso adelante en llevar el concepto real de conversación al ámbito computacional. Una ontología es una colección de conceptos, predicados, secuencias, términos y relaciones entre estos elementos, que son entendibles por una sociedad de agentes. Para comunicarse los agentes realizan intervenciones, traducidas en mensajes compuestos por un sobre y un contenido, el sobre especifica los datos del agente emisor, datos de los agentes receptores, y los datos de la intervención, tales como el lenguaje, ontología y protocolo utilizado. El contenido incluye dos partes: una performativa que indica la acción general del mensaje, y una frase compuesta por elementos pertenecientes a la ontología usada, que indica sobre que aspectos habla la performativa. Cada agente comprende esta intervención y en consecuencia, afecta su estado interno, y eventualmente reaccionará con un conjunto de intervenciones. Este aspecto comunicacional es actualmente tema de estudio por computistas, ingenieros, y teóricos del lenguaje, ya que es un campo que puede dar aportes significativos a la ciencia de la computación.

Una aspecto básico que debe ser modelado de forma transparente es el medio, tecnología o protocolo utilizado para implementar la comunicación. Esta implementación puede ser de distinto tipo, se puede usar mensajería entre procesos, protocolos para objetos distribuidos (por ejemplo RMI:Remote Method Invocation, DCOM:Distributed Component Model o CORBA), servicios Web, tecnologías para la internet, protocolos como el HTTP(HyperText Transport Protocol), o comunicación vía sockets[3,20]. Lo importante es que la comunicación, para que sea efectiva, cumpla con ciertas características inherentes

a los agentes, entre las cuales se pueden nombrar la asincronía y el uso de ontologías.

Existe otro tipo de comunicación, que es indirecta y que no está basada en el pase de mensajes. Este tipo de comunicación deriva de las investigaciones en inteligencia artificial, específicamente de las colonias de hormigas, está basada en variables compartidas por todos los agentes y por mediciones del ambiente donde estos se desenvuelven.

1.1.3. Sociabilidad:

Esta capacidad está muy relacionada con el aspecto comunicacional, ya que la comunicación juega un papel importante en la eficiencia de las sociedades de agentes. Una sociedad de agentes es un grupo de agente que interactúan, se comunican, conversan, "piensan" y actúan en conjunto para lograr un objetivo común. En este sentido, varios autores han sugerido un conjunto de protocolos basados en esquemas utilizados por las sociedades de humanos, por ejemplo, uno de lo más populares es el Contract-net[7][2], que tiene como finalidad establecer una relación de contrato entre dos o más agentes. Este protocolo consiste en que un agente "iniciador" envía un mensaje a un grupo de agentes demandando un servicio, cada agente receptor puede enviar una propuesta o un mensaje rechazando la demanda del servicio. El agente iniciador evalúa cada una de las propuestas y escoge un ganador, con el cuál se establece un contrato de servicio. Así como este protocolo, existen otros orientados a subastas, bolsas bursátiles y otras actividades de la vida humana, que intentan imitar conductas grupales que llevan al logro de objetivos específicos.

1.1.4. Reactividad:

La capacidad de emitir una acción inmediata al recibir una señal o percibir un estado en el ambiente, es lo que caracteriza a los agentes reactivos. Los agentes, por lo general, no reaccionan de inmediato, ya que deben procesar la

[2] El protocolo *Contract-net* es uno de los protocolos establecidos por la FIPA (Foundation Intelligent Phisical Agent) para la coordinación de agentes: ver http://www.fipa.org.

información y “pensar” sus acciones. La reactividad en los agentes posibilita acciones rápidas, cruciales en sistemas de tiempo real que simplemente, no ameritan aplicar reglas complejas.

1.1.5. Inteligencia:

Generalmente, la cualidad de inteligencia es asociada directamente con el concepto de agente. Debido a que un agente debe analizar y tomar una acción de forma autónoma, es necesario implementar esta característica utilizando alguna tecnología o técnica computacional, para lo cual generalmente se utilizan técnicas inteligentes.

Los sistemas expertos es la técnica predilecta para imprimir inteligencia a los agentes, ya que permiten a través de un conjunto de reglas finitas, las cuales pueden estar asociadas a variables del ambiente, llegar a una conclusión que genere o inhibe una determinada acción. Esto encaja con la noción de inteligencia que se le atribuye a los agentes; pero no es la única técnica que puede ser utilizada cuando se diseñan sistemas multiagentes. El paradigma basado en agentes nace como una extensión de la inteligencia artificial, y no como un caso particular de los sistemas expertos. Se pueden utilizar un conjunto variado de técnicas inteligentes en la construcción de agentes, entre las cuales se pueden nombrar a las reglas difusas para analizar situaciones dinámicas, las redes neuronales para predecir comportamientos y variables del ambiente, las colonias de hormigas como una técnica de coordinación entre agentes, los algoritmos genéticos como método de búsqueda, entre otros. En la medida en que se integren más técnicas inteligentes a la construcción de agentes, se logrará un acercamiento más certero a la característica inteligente de los agentes.

1.1.6. Movilidad:

Es la capacidad que tiene un agente de mover su estado y código de ejecución de un nodo a otro en un sistema distribuido. Esta capacidad posibilita una computación menos centralizada y más distribuida. Un agente puede alojarse

en cualquier nodo y realizar sus tareas utilizando los recursos locales, para después volver a su nodo origen llevando la información procesada (ver figura 1).

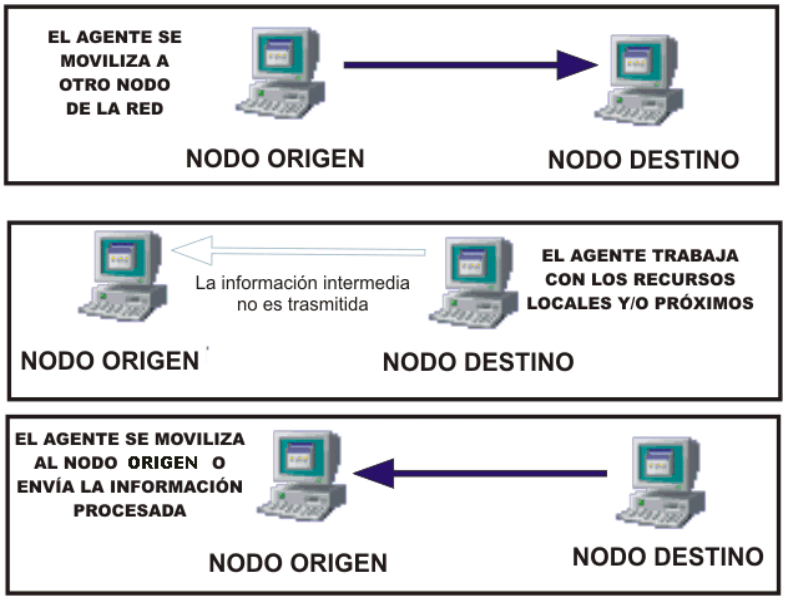


Figura 1. Agentes Móviles

Este tipo de comportamiento ahorra ancho de banda, ya que no se trasmite la información intermedia que se genera cuando se solicita un servicio remoto. Otra ventaja de los agentes móviles es que pueden moverse por una red pequeña, mediana o incluso a través de internet visitando nodos distantes geográficamente. También, pueden realizar tareas difíciles de coordinar si se utiliza un esquema centralizado, donde todas las acciones dependen de un único nodo donde se consumen la gran parte de los recursos.

Las anteriores características describen en cierta medida los atributos de los agentes. Pero existen otras cualidades que se les atribuyen, tales como cooperación, colaboración, competencia, pro-actividad, entre otras, pero que se consideran muy relacionadas con las explicadas anteriormente. Por ejemplo, la colaboración y la competencia son esquemas de la sociabilidad de los agentes, y la pro actividad está estrechamente ligada con la autonomía de los agentes.

1.2. Arquitectura de los Agentes

No existe una única arquitectura para diseñar y construir agentes;. Por ejemplo, una corriente se ha inclinado por el diseño de agentes utilizando programación lógica, lo que supone una estrecha vinculación a un motor de inferencia con colecciones de hechos; otra corriente ha vinculado los agentes con los sistemas de control, y también, existen arquitecturas específicas basadas en creencias y deseos. En un ejercicio de categorizar las diferentes visiones que existen actualmente con respecto a este tema, a continuación se describen algunas de éstas arquitecturas.

1.2.1. Basada en lógica

Este tipo de arquitectura surgió como una extensión directa de las investigaciones que en el área de lógica computacional se han realizado en los últimos años. Sugiere que la cualidad inteligente de los agentes puede ser construida utilizando un lenguaje basado en lógica. Estos lenguajes permiten representaciones simbólicas de hechos reales, y su manipulación sintáctica es hecha utilizando lógica de primer orden. Bajo este tipo de arquitectura cada agente procede a ejecutar acciones después de realizar un proceso deductivo, basado en una base de datos que hace el papel de las creencias humanas.

1.2.2. Arquitectura reactiva

Se basan en una visión orientada a la percepción-control-acción. Los agentes en su definición deben ser capaces de percibir estados del ambiente, y actuar en función de ellos. Este tipo de arquitectura no considera una base de datos con experiencia, creencias o hechos; solo se realiza la acción observando el estado del ambiente para proceder a modificarlo con la finalidad de llevar el entorno a un estado controlado.

1.2.3. Arquitectura Creencia-Deseo-Intención

Tienen su fundamento en el razonamiento práctico humano, que actúa cada vez que producimos una decisión. Esto involucra los objetivos que se

desean conseguir, las opciones que están disponibles, el conocimiento sobre el entorno, y las acciones que se deben ejecutar para lograr los objetivos propuestos. Podemos describir los componentes de esta arquitectura de la siguiente forma:

Creencias: es la base de datos de hechos y reglas que cada agente contiene. Es la base de conocimiento sobre el entorno en que se desenvuelve el agente. Las creencias en conjunto con la entrada del ambiente son procesadas con la finalidad de tomar de decisiones.

Deseos: se definen como las metas finales a alcanzar. En función de ellos se generan las opciones, alternativas o comportamientos a seguir para lograr los objetivos propuestos. Utilizando la base de creencias se escoge la opción que se considera más acertada.

Intenciones: es el conjunto de acciones sin ejecutar como producto del sistema de creencias y deseos del agente.

Weiss escribe un macro-algoritmo de esta arquitectura que ilustra su funcionamiento:


```
function action(p: P) :A
begin
1.    B' := brf(B,p)
2.    D' :=options(D,I)
3.    I' :=filter(B',D',I)
return execute (I)
end function action
```

Este algoritmo se inicia ejecutando la función *brf* (función de revisión de creencias), que toma como entrada el conjunto de creencias generales (B) y algunas variables del ambiente (p); la función retorna un subconjunto de B (B'), asociados con la entrada p. En la segunda línea se generan las opciones, la función *options* toma como entrada los deseos (D) e intenciones generales del agente generando un subconjunto de deseos (D') asociados con los objetivos específicos que se desean lograr. La tercera línea aplica un filtro al conjunto de intenciones generales, desechando las que no concuerdan ni con los deseos específicos, ni con las creencias específicas generadas dentro de la función *action*. La función *filter* retorna el conjunto de intenciones que pasaron el filtro (i').

Finalmente para generar la acción deben ejecutarse el conjunto de intenciones obtenidas.

1.2.4. Arquitectura por niveles

Esta arquitectura es la combinación de las otras arquitecturas propuestas. En la mayoría de los casos los agentes deben poseer un conjunto de cualidades que no pertenecen a una única arquitectura. Por ejemplo, deben ser reactivos algunas veces, y en otras autónomos y pro-activos, o también, pueden poseer un módulo basado en lógica que esté integrado con una arquitectura BDI. Es por esta razón que varios autores han propuesto una arquitectura que integre las características de cada una de las otras arquitecturas, y que se distribuyan en forma de niveles. Los niveles pueden estar acoplados en dos posiciones: vertical, donde se obtiene una única salida, y la entrada es procesada por varios niveles de forma secuencial; u horizontal, donde la entrada es procesada en forma paralela por los diferentes niveles y se obtienen diferentes propuestas de acciones.

Existen otras arquitecturas que son casos particulares de la arquitectura por niveles, y que están documentadas ampliamente en la literatura.

1.3. Modelo de estados de agentes

El modelo tratado en este trabajo, define un conjunto de estados, que permiten establecer y conocer la situación en la cual se encuentra el agente instanciado en su ciclo de vida. Los estados posibles son:

Inicializado: el agente ha sido creado, no ha sido registrado en el administrador de agentes, no se le ha asignado un nombre ni puede comunicarse con otros agentes.

Activo: el agente está registrado, se le ha asignado una dirección o nombre con el cuál puede ser localizado.

Suspendido: la ejecución del comportamiento del agente ha sido suspendido, el agente ha sido detenido.

Eliminado: el agente ha concluido su ciclo de vida, la ejecución de su comportamiento ha finalizado y ya no está registrado en el Administrador de agentes.

En espera: el agente está bloqueado en espera de un evento o condición.

En tránsito: el agente está siendo migrado hacia otro contenedor y/o plataforma.

En Copia: el agente está siendo copiado o en proceso de clonación.

En llegada: el agente está siendo instalado en un nuevo contenedor o plataforma y se encuentra en un estado estable.

El modelo de clases de agentes define los cambios de estado según el ciclo de vida del agente y las acciones que sobre éste se realicen.

1.4. El modelo de comportamientos de agentes

En esta sección se muestra un modelo de comportamientos utilizado en el diseño del SGS. El modelo permite definir un esquema basado en comportamientos reutilizables para cada agente. Se debe definir una configuración de tareas, que no es más que las secuencias entre las tareas, y tipos de transiciones entre ellas. Los tipos de comportamientos derivan de una clase abstracta llamada *Behaviour*. Estos tipos están definidos según el diagrama de transiciones entre tareas especificado. Las transiciones entre tareas tienen como característica que pueden estar bloqueadas en espera de un evento o condición. Los tipos de comportamientos disponibles se muestran en la tabla 1.

Nombre	Descripción
Behaviour	Es la clase base del modelo de comportamientos, posee los métodos necesarios para la planificación de tareas y las rutinas necesarias para ejecutar transiciones entre estados: iniciar, reiniciar y bloquear.
SimpleBehaviour	Es una clase abstracta de la que derivan los comportamientos de una sola tarea, o también llamados comportamientos atómicos.
OneShotBehaviour	Es una clase abstracta que se utiliza para definir

	comportamientos atómicos, que solo se pueden ejecutar una vez, y que no pueden estar bloqueados.
CyclicBehaviour	Es una clase abstracta que se usa para definir comportamientos cíclicos o cuya secuencia de tareas se repite indefinidamente.
CompositeBehaviour	Es una clase abstracta que se utiliza para definir comportamientos que están compuestos de otros comportamientos. En esta clase no se define políticas de planificación de tareas.
SequentialBehaviour	Es una clase abstracta de la que derivan comportamientos que tienen una secuencia definida o un único camino de estados de tareas a ejecutar.
ParallelBehaviour	Esta clase deriva de *CompositeBehaviour*, y define una política de transiciones para los sub-comportamientos. Se ejecutan los sub-comportamientos de forma concurrente y termina el comportamiento padre cuando todos los sub-comportamientos han llegado a su condición de finalización.
FSMBehaviour	Esta clase deriva de *CompositeBehaviour*, define un comportamiento basado en una máquina de estados definida por el usuario.
SenderBehaviour	Esta clase implementa un comportamiento atómico que ejecuta la acción de enviar un mensaje a otro agente de la plataforma.
ReceiverBehaviour	Esta clase implementa un comportamiento atómico que ejecuta la acción de recibir un mensaje desde otro agente de la plataforma.
WakerBehaviour	Es una clase abstracta de la que derivan comportamientos atómicos que esperan un tiempo pre-definido para ser ejecutados.

Tabla 1. Lista de Comportamientos

El modelo de clases para el esquema de comportamientos se constituye en una jerarquía de clases (ver figura 2). Este modelo dibuja un árbol de clases donde las clases hijas derivan las propiedades y métodos de la clase padre. Todas las clases de la jerarquía poseen los métodos definidos en la clase *Behaviour*. El método *action* (que puede ser sobrescrito) define el conjunto de acciones que definen un tipo de comportamiento, el método done indica si el comportamiento todavía está siendo ejecutado o ha concluido. Los métodos *onStart*, *onEnd*, *block* y *restart* están asociados con las transiciones entre tareas.

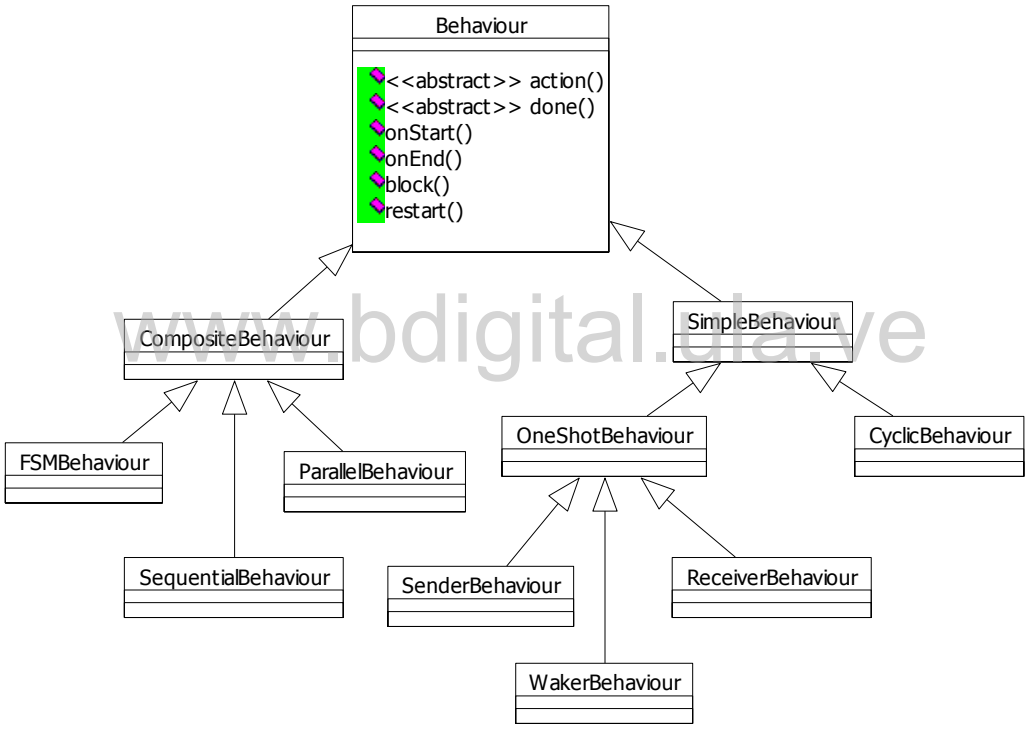


Figura 2. Modelo de clases para comportamientos.

1.5. El modelo de comunicación

El modelo de comunicación tratado en este trabajo, puede ser descrito en dos fases, la primera referente al esquema de la plataforma donde opera, y la segunda referente a los protocolos que utiliza para la comunicación entre agentes.

Se ofrece un entorno multiplataforma, orientado a objetos, que implica gran capacidad de abstracción ofreciendo al mismo tiempo interoperabilidad,

confiabilidad y seguridad. El esquema de comunicación indica que en cada nodo de red que opere con el protocolo TCP-IP, se puede definir una plataforma donde están registrados un conjunto de contenedores, en estos contenedores se encuentran registrados instancias de agentes, cada uno de ellos posee una dirección única similar a la que adjudica el protocolo que se utiliza para envío y recepción de correo electrónico (SMTP). El protocolo usado para especificar nombres se define de la siguiente manera: un nombre de agente seguido por el carácter arroba (@), seguido del nombre de la plataforma, que a su vez está compuesta del nombre del nodo de red y del nombre del contenedor, por ejemplo: agente1@localhost:1099/JADE. Esta propiedad de direccionamiento permite contar con un sistema de nombramiento que es utilizado por los demás agentes del entorno multi-agentes para establecer relaciones de comunicación.

La descripción de la segunda fase empieza con explicar el modelo orientado a mensajes que sigue las especificaciones FIPA[5,6,7,8,9]. Los mensajes utilizan un lenguaje ACL (Lenguaje de comunicación para Agentes) para ser expresados. El mensaje está compuesto por dos partes, un envoltorio o sobre donde se escriben propiedades tales como el agente remitente, el agente receptor, y el protocolo de comunicación a utilizar; en este sentido, es importante señalar, que cuando se envía un mensaje solo es necesario especificar los atributos antes descritos para que el mensaje llegue a su destino lo que otorga la propiedades de transparencia y nombramiento al modelo de comunicación. La segunda parte del mensaje es el contenido, que puede estar expresado en un texto plano, o en un texto que siga una ontología predefinida o alguna ontología definida por el usuario.

1.6. Metodología MAS-CommonKads

Es una extensión de la metodología CommonKads[13], con el agregado que se específica para sistemas multiagentes (ver figura 3). El prefijo MAS que inglés significa MultiAgent Systems, es incluido para distinguirla de la metodología general. Esta metodología se basa en un meta-modelo que describe y recoge las características básicas de los sistemas multiagentes; y ofrece un conjunto de modelos que agrupan los elementos básicos y necesarios para describir al SMA,

tales como las tareas, agentes, y las comunicaciones. Los modelos están basados en la identificación de atributos y sus relaciones; el producto generado consiste en una serie de plantillas que describen al sistema multiagente. Son modelos que implican un desarrollo cíclico, haciendo uso de la ingeniería de software.

1.6.1. Modelos de la metodología

La metodología debe generar los siguientes modelos:

Modelo de agente (AM): un agente es un ejecutor de una determinada tarea. Bajo esta metodología un agente puede ser un humano, un sistema de software o cualquier otra entidad capaz de ejecutar una tarea. Este modelo describe las características y capacidades de los agentes del sistema.

Modelo de tareas (TM): describe las tareas que se deben realizar en el entorno organizacional. También sirve como referencia para distribuir las tareas entre los agentes.

Modelo de organización (OM): es una herramienta para describir la organización donde se integrará el sistema multiagente. Se utilizan técnicas asociadas con el desarrollo de sistemas basados en conocimientos (en sus siglas en inglés KBS knowledge based systems).

Modelo de comunicación (CM): detalla las intervenciones y conversaciones que tienen que darse entre los agentes para realizar las tareas propuestas en el modelo de tareas. Como herramienta se propone el uso del modelo UML.

Modelo de experticia (EM): está dividido en tres subniveles: nivel de dominio, conocimiento declarativo sobre el dominio; nivel de inferencia, una librería de estructuras de inferencias para ser usadas por los agentes, y el nivel de tareas, donde se encuentran las tareas ejecutadas por el motor de inferencia.

Modelo de diseño (DM): describe la plataforma computacional y el diseño técnico con la finalidad de realizar el proceso de implementación.

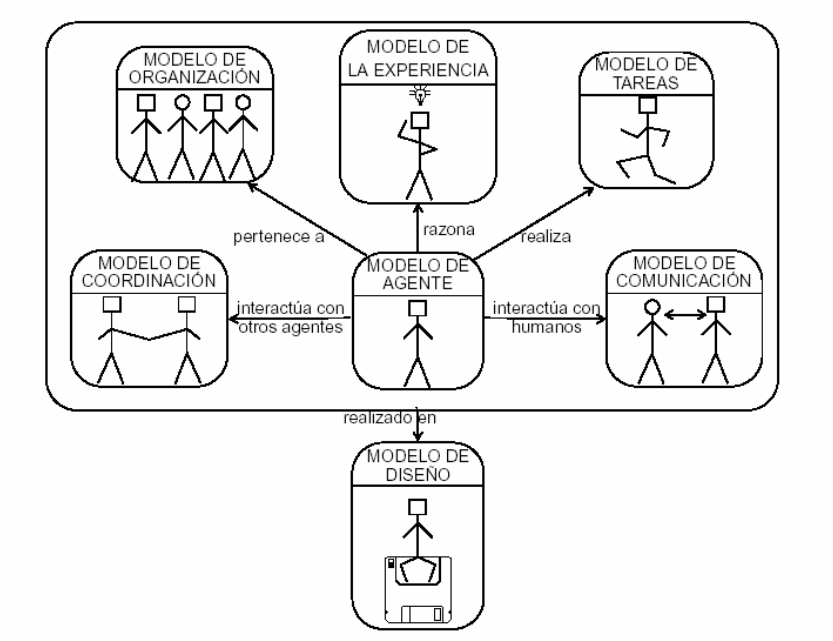


Figura 3. Modelo MAS-CommonKads[13]

1.6.2. Ciclo de desarrollo del MAS-CommonKads

Conceptualización: enumeración de tareas para obtener la primera visión del problema, se determinan los casos de uso que servirán para entender los requerimientos iniciales.

Análisis: Esta parte en muchas metodologías se conoce como análisis de requerimientos, ya que su objetivo o producto es la lista de requerimientos formales para la construcción del SMA. Se escogen la tecnología, y herramientas a utilizar que cumplan con los requerimientos.

Diseño: en esta parte se estudian los requerimientos y se construye la arquitectura del sistema en función de ellos. Se hacen modelos detallados los partes que formarán el sistema.

Codificación y prueba: en esta sección se codifica cada agente utilizando las herramientas escogidas.

Integración: se realiza la integración con la plataforma.

Operación y mantenimiento.

1.6.3. Extensión del modelo MAS-CommonKads

En el marco del proyecto Agenda Petróleo, se realizó una extensión a la metodología MAS-CommonKads[13,14]. Se redefinieron los modelos de coordinación e inteligencia, y se propuso una extensión en los modelos de tareas, agentes y comunicación.

El modelo de inteligencia (ver figura 4), está compuesto por un conjunto de elementos que está en correspondencia con el razonamiento humano. Se propone un esquema que sigue los parámetros generales de la metodología y que integra conceptos como experiencia, dominio, conocimiento estratégico, aprendizaje y razonamiento.

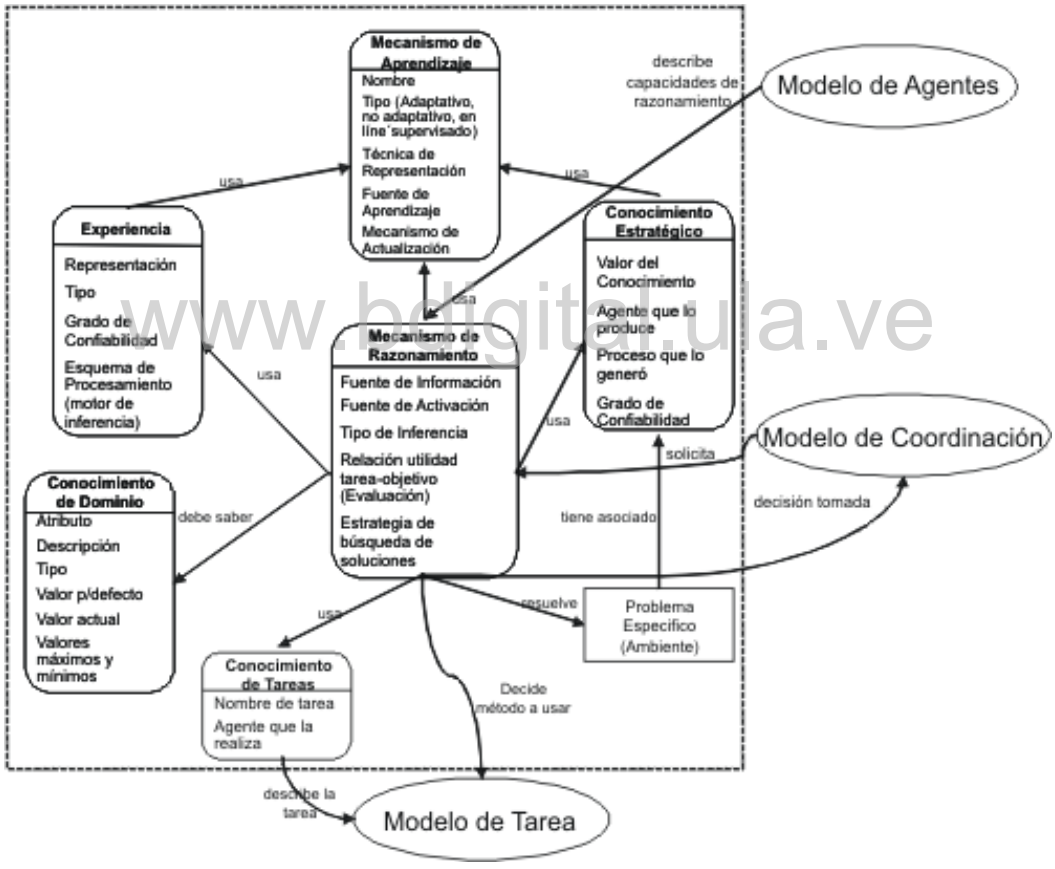


Figura 4. Modelo de Inteligencia

La figura 4 muestra el modelo de inteligencia con los atributos por cada componente, sus relaciones internas y su interrelación con los demás modelos de la metodología. La inteligencia de los agentes está relacionada con las tareas, ya que a partir del mecanismo de razonamiento se escogen y deciden el conjunto de

tareas que se deben ejecutar para la resolución de un problema. El componente inteligente modelado se percibe como un módulo que puede ser insertado y extraído de la estructura de cada agente, agregándole de esta forma capacidades inteligentes, sin que esto comprometa la arquitectura de los agentes.

El modelo de coordinación propuesto pretende que el usuario defina los esquemas de intercambio de comunicación necesarios para el logro de las metas grupales. El modelo está basado en que se pueden establecer dos tipos de comunicación entre agentes: una directa basada en pase de mensajes y uso de ontologías, y otra indirecta basada en estrategias de memoria compartida y de métodos de estímulo-respuesta (figura 5).

El modelo de coordinación permite al usuario establecer el conjunto de estrategias que las comunidades de agentes usarán para el logro de los objetivos grupales. Estas estrategias, por lo general, imitan actitudes de coordinación de grupos humanos, tales como los procesos de contratación, de subastas, etc., donde la interacción y el intercambio de información generan acciones individuales que sumadas permiten alcanzar metas específicas.

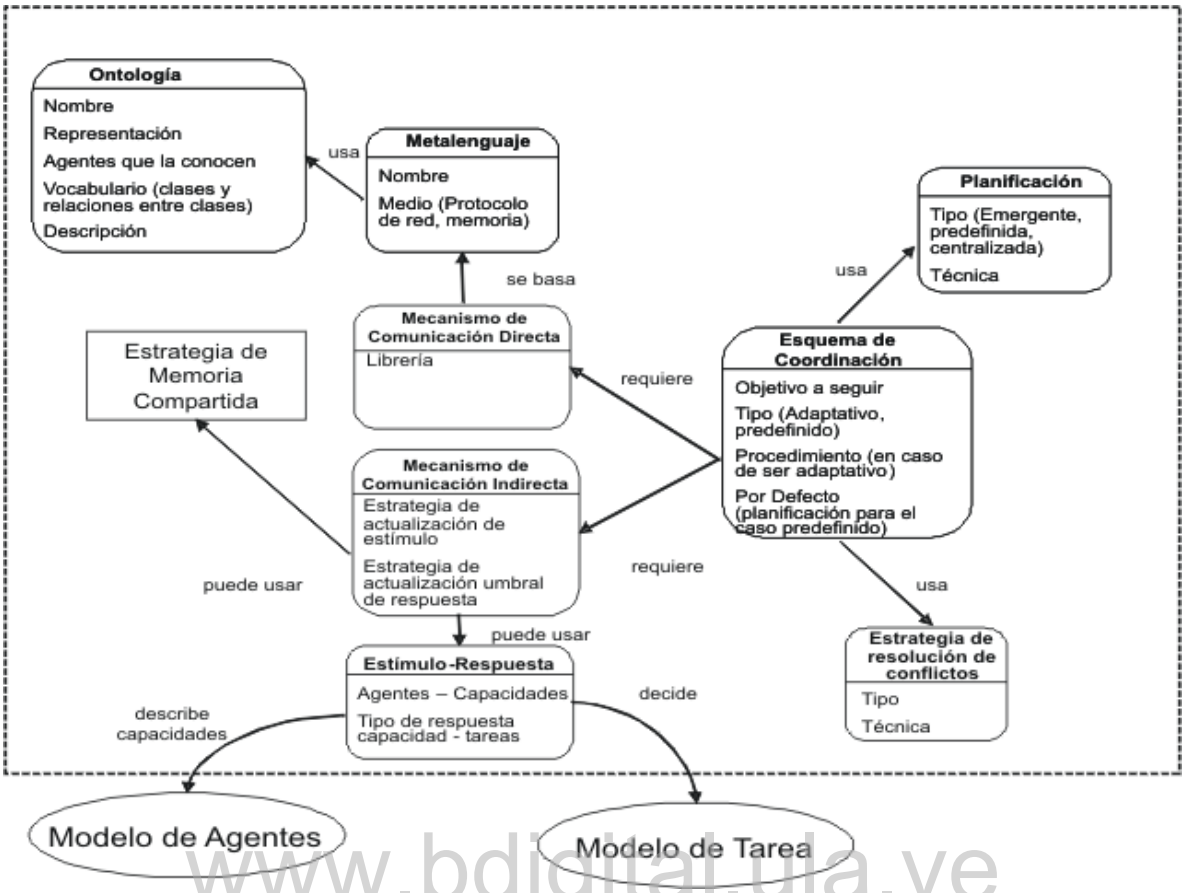


Figura 5. Modelo de coordinación

Los atributos de cada componente del modelo de coordinación (ver figura 5) permiten especificar el tipo de planificación, el tipo de comunicación con sus detalles, y finalmente las estrategias de resolución de conflictos.

2. Sistemas distribuidos

Cuando se desarrollan sistemas multiagentes, es importante contar con una plataforma distribuida que brinde un conjunto de servicios para las sociedades de agentes. Es por esto que esta capa constituye una parte esencial del sistema, debe garantizar transparencia y seguridad en las transacciones, interoperabilidad de las aplicaciones y componentes de software, migración de agentes, objetos y/o recursos, comunicación ínter-procesos, y proveer de un sistema de nombramiento para la localización de agentes y/o objetos. Esta capa distribuida garantiza que el sistema multiagente cuente con propiedades de comunicación y coordinación, esenciales para el logro de los objetivos de la comunidad de agentes.

2.1. Características de los sistemas distribuidos asociadas con sistemas multiagentes

Las características de los sistemas distribuidos que sirven de base para la construcción de sistemas multiagentes, se presentan a continuación:

Transparencia: se refiere al manejo de servicios siguiendo la filosofía de ocultar detalles de "cómo" se provee el servicio concentrándose en "qué" provee el servicio, es decir, al usuario se le ocultan detalles sobre la ejecución del servicio que no se consideran relevantes. El sistema se encarga de proveer el servicio manejando en una capa oculta los detalles de implementación.

Seguridad: debido a que el esquema distribuido opera bajo un sistema en red y de acceso externo, es necesario implementar mecanismos para mantener un nivel de seguridad que imposibiliten el acceso a aplicaciones y a datos por una fuente no acreditada. En el mercado existen diferentes y variadas aplicaciones que implementan estos mecanismos, por ejemplo entre estos mecanismos podemos citar los algoritmos que utilizan criptografía, los algoritmos de validación de contraseñas, y los algoritmos de clave pública y privada, entre otros.

Interoperabilidad: un problema recurrente en sistemas computacionales en red es la de integrar diferentes plataformas de software, y diferentes aplicaciones corriendo en sistemas operativos distintos que deben cumplir una función en conjunto. La aparición de nuevos lenguajes de programación que implementan máquinas virtuales o código 100% portable, sistemas basados en objetos como CORBA, y lenguajes de marcas como XML (eXtended Markup Language), han dado una respuesta a este problema.

Migración: en sistemas basados en tecnología Java o CORBA, u otra tecnología similar, es posible implantar agentes u objetos que migran. Existen varias técnicas y modos de migración para agentes y objetos. Para el caso de los agentes, que en la mayoría de los casos, son extensiones de objetos, es necesario contar con

una plataforma comunicacional con ciertas características. Los agentes móviles, generalmente se utilizan para recolectar información de varios servidores; se mueven o migran utilizando un protocolo común y establecen sus operaciones localmente. Los objetos se hospedan y ejecutan sus métodos en servidores, y en algunos casos pueden desplazarse de una localidad a otra de la red. En el caso de los sistemas multiagentes se proveen canales que permiten que los agentes viajen a través de los diferentes nodos de la red. En este tipo de ambiente un elemento importante a proveer es la seguridad, ya que una incursión de un agente o software externo puede reportar graves daños.

Mensajería y comunicación interprocesos: es posible comunicar aplicaciones que corren en localidades diferentes a través de sistemas de mensajerías que pueden ir de niveles bajos a altos niveles de abstracción. Las aplicaciones pueden comunicarse entre ellas a bajo nivel, por ejemplo, a través de *sockets* (bajo protocolo TCP/IP), y a niveles más altos con sistemas de mensajería como XML o SOAP (Simple Object Access Protocol). También existen otros mecanismos de comunicación que son provistos por sistemas o plataformas como Java o CORBA. Finalmente, sistema de pases de mensajes o de pizarrón (cada agente escribe en una archivo o "pizarra" común) también son aplicables. Todos estos sistemas de mensajería sirven de apoyo a los lenguajes provistos para comunicación entre los agentes.

Sistema de Nombramiento: para poder localizar eficientemente agentes, objetos y/o recursos es necesario contar con un sistema de nombramiento confiable, que indique la función y pertenencia del agente u objeto en cuestión. También debe permitir acceder y manipular estos objetos de forma no ambigua. El paradigma multiagente toma el concepto de "nombramiento" de los sistemas distribuidos y lo lleva al plano de los agentes. FIPA: Foundation for Intelligent Phisical Agent[5,6,7,8,9]. una fundación conformada por un conjunto de organizaciones mundialmente reconocidas asociadas al área tecnológica, expresa en sus especificaciones sobre sistemas multiagentes, que en estos sistemas debe existir

un directorio de agentes, también llamado “páginas blancas”, el cuál debe ser administrado por un agente coordinador, quién es el encargado de ejercer el control sobre el sistema multiagente. este agente tiene la potestad de crear, iniciar, suspender y autorizar migraciones de todos los agentes. Los sistemas de nombramiento se implantan a través del uso de una base de datos que asocia (*bind*) recursos con nombres descriptivos, a los cuales se les puede asociar una jerarquía. Uno de los ejemplos de sistemas de nombramiento es el DNS (Domain Naming Server); sistema encargado de dirigir el nombramiento en la internet y que implementa “espacios de nombres” (namespace) o “dominios de nombres”, para soportar dominios de localidad por países, por función y por tipo de servicio.

2.2. Acceso a recursos mediante el uso componentes

Los componentes son piezas de software reutilizables, disponibles para diferentes lenguajes y aplicaciones a través de una interfaz común. Permiten encapsular el acceso a datos, la lógica de negocio, el acceso a recursos de hardware e interfaces con aplicaciones. Se basan en una extensión de los objetos, tienen métodos y propiedades, y permiten persistencia, acceso remoto y un modelo distribuido.

En la actualidad existen varios esquemas que plantean una visión orientada a componentes Los dos esquemas más utilizados son el COM (Component Object Model) [22] propuestos para sistemas bajo sistemas operativos Windows, y el modelo basado en Beans propuesto por Sun Microsystems para arquitecturas basadas en tecnología Java.

2.2.1. Modelo Objeto-Componente (COM Component Object Model)

Este modelo propuesto para sistemas Windows, permite encapsular recursos de datos, de lógica de negocio, o de otro tipo en un componente que puede ser accedido por distintos lenguajes, aplicaciones y entornos de desarrollo de software. Es muy popular ya que está disponible para la mayoría de los lenguajes bajo plataforma Windows, además de poder usarlo en aplicaciones como procesadores de palabras, hojas de cálculo, sistemas de bases de datos en

tiempo real, etc. COM es utilizado actualmente por más de 150 millones de sistemas de software en el mundo [22].

También es posible construir e instalar objetos COM que pueden ser utilizados por terceros, solo es necesario cumplir con una serie de pasos, que incluyen generar el envoltorio para la aplicación o recurso que se quiere publicar. Los envoltorios COM pueden ser accedidos desde aplicaciones, sistemas y lenguajes de programación, de tal forma que se mantenga una interfaz común. Esta interfaz es generada desde un lenguaje IDL (Interface Definition Language) que permite especificar un conjunto de cabeceras que sirven como intermediario entre el sistema cliente y una implementación escrita en un lenguaje determinado. Cuando se tiene una biblioteca de funciones y se quiere que estén disponibles para alguna aplicación de escritorio, de diseño gráfico o arquitectónico, algún sistema de publicación web o lenguajes script, o algún otro lenguaje en particular, es posible "envolver" estas funciones y exportarla a través del uso de la interfaz común COM.

Existe una extensión de este modelo denominada DCOM (Distributed Component Object Model) que permite acceder a los recursos de forma remota, además de la posibilidad de distribuir la carga entre un conjunto de componentes. Este modelo es bastante utilizado ya que es accesible desde casi cualquier lenguaje que opere bajo sistemas operativos Windows, aunque no tiene una separación visible entre la lógica de negocio y el acceso a datos, la mayoría de fabricantes de entornos de desarrollo de software le dan soporte.

2.2.2. Modelo basado en Beans

Un *JavaBean*[16,20] es una pieza de software que puede ser usado varias veces, la cuál está escrita en Java. Este concepto coincide con el concepto de componente. Este modelo propuesto para el lenguaje Java, además de heredar todas las ventajas y desventajas del entorno de programación, provee una visión clara para aplicaciones multicapa (ver figura 6). Está bien separado el concepto de capa de lógica del negocio que agrupa los componentes *SessionBean* y el

concepto de capa de acceso a recursos y datos que agrupa los componentes del tipo *EntityBean*.

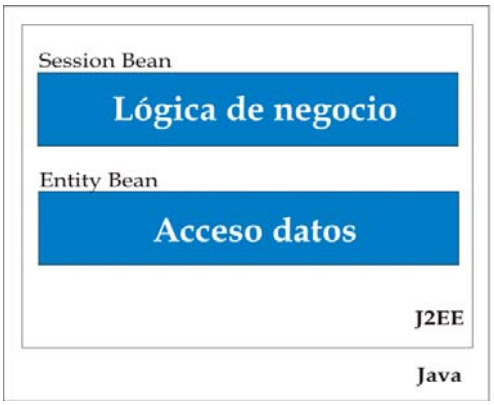


Figura 6. Modelo de Componentes JavaBean

Este modelo debe su popularidad a que forma parte del J2EE (Java 2 Enterprise Edition)[22], que permite manejar las aplicaciones de servidor (back-end) bajo un modelo coherente, distribuido, y orientado a objetos. La característica multiplataforma heredada de Java, le permite integrar servicios y aplicaciones en entornos heterogéneos.

Una desventaja de este modelo es que solo permite crear componentes desde el lenguaje Java, no es posible hacer uso de otro lenguaje, lo que restringe la comunidad de programadores que desarrollan aplicaciones con esta herramienta. Debido a sus características los componentes *JavaBean*, no son muy utilizados en el área de aplicaciones de escritorio, sus usos están orientados al manejo de aplicaciones del tipo back-end (servidor) como manejadores de bases de datos, servidores de aplicaciones, etc.

CAPITULO II: Sistema de Control Distribuido Inteligente basado en Agentes (SCDIA)

En una plataforma de automatización para control de procesos industriales (ver figura 7) existen varios niveles caracterizados cada uno de ellos por la granularidad de la información que manejan y por los objetivos que deben cumplir. Estos niveles deben compartir recursos e interrelacionarse.

Figura 7. Modelo de plataforma de automatización ISO/OSI

Este trabajo forma parte del SCDIA (Sistema de Control Distribuido basado en agentes)[1,2,13,18]. El SCDIA debe funcionar como una poderosa herramienta para resolver problemas complejos de control y optimizar procesos en la industria. La necesidad de integrar aplicaciones heterogéneas, aplicaciones legadas, que deben cooperar compartiendo datos e información y que funcionan con un mismo objetivo, es una característica recurrente en sistemas de redes de computadoras empresariales e industriales. El SCDIA hace uso de los nuevos paradigmas en computación, permitiendo que el control de los procesos surja de las interacciones y del conocimiento implícito y explícito en los distintos componentes del sistema de control.

1. El Sistema de Control Distribuido basado en Agentes (SCDIA)

El SCDIA es un modelo para la implementación de sistemas multiagentes en plataformas distribuidas y heterogéneas dedicadas al control y automatización

de procesos industriales. Este modelo describe cinco tipos de agentes configurados para tareas de alto nivel, asociadas a la coordinación, control, medición y a tareas especializadas en la plataforma de automatización[14,18].

Los tipos de agentes descritos por el modelo SCDIA son los siguientes:

Agentes del SCDIA

1. Agente Coordinador
2. Agente Controlador
3. Agente Especializado
4. Agente de Medición
5. Agente de Actuación

Agentes del Sistema de Gestión de Servicios

6. Agente Administrador de Agentes
7. Agente Gestor de Recursos
8. Agente Gestor de Aplicaciones
9. Agente de Base de Datos
10. Agente de Control de Comunicación

Estos tipos permiten construir comunidades de agentes coordinadas dedicadas a lograr metas y objetivos específicos dentro de la organización.

El SCDIA también describe modelos de coordinación, inteligencia, experiencia y colaboración que guían el proceso de construcción de sistemas multiagentes (SMA).

El SCDIA está pensado para sistemas de automatización industrial, donde la información tiene un flujo específico correspondiente a este tipo de organización. La arquitectura propone una comunidad de agentes que representan los componentes de un lazo de control de procesos genérico, donde las actividades de cada agente están relacionadas y se consolidan para el logro en un objetivo común. La arquitectura de la plataforma propone cinco tipos de agentes:

1. Agente de Medición

Este agente funciona como una interfaz para el acceso a la información que se genera en la capa del proceso. Los agentes de SCDIA interactúan con este agente para obtener información de la capa de proceso.

2. Agente Controlador

Es el encargado de procesar la información y tomar las decisiones en el lazo de control. Sus políticas están basadas en la teoría de control de procesos.

Agente de actuación

Es el ejecutor de las decisiones generadas por los agentes controladores y coordinadores. Traduce estas decisiones en acciones específicas sobre el ambiente.

3. Agente Coordinador

Es el agente supervisor del estado de los procesos llevados a cabo en el lazo de control, planifica y toma de decisiones para las comunidades de agentes.

4. Agente Especializado

Es el agente de realizar tareas específicas dentro de la plataforma. Tiene un objetivo determinado que debe ejecutar según ciertas condiciones.

Fígura 8. Arquitectura del SCDIA

Esta arquitectura se identifica con las actividades realizadas en un proceso controlado automáticamente, pero el esquema puede ser extendido hacia toda la jerarquía de automatización. Los agentes del SCDIA llevan a cabo tareas de diferente tipo, dependiendo el nivel dentro de la jerarquía donde se encuentren.

2. Sistema de Gestión de Servicios (SGS)

Como apoyo a la plataforma SCDIA se desarrolla un SGS (Middleware) constituido por componentes de software en un ambiente distribuido y heterogéneo Cada componente puede actuar como vía de acceso al procesamiento para una determinada aplicación, como puente entre clientes remotos y fuentes de datos, o interfaz de acceso a recursos y sistemas de información. El *middleware* es el corazón del sistema de agentes distribuido, puesto que en él moran los agentes que manejan los servicios de comunicación y le otorgan al sistema de agentes características tales como seguridad, transparencia, nombramiento, migración e interoperabilidad. El modelo del SGS está estructurado en tres niveles (ver figura 8).

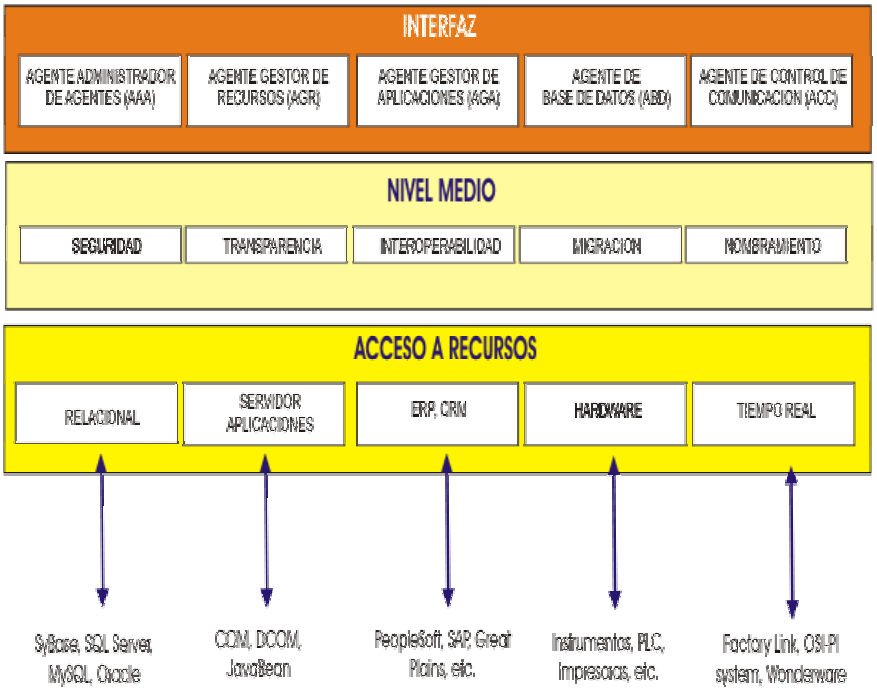


Figura 8. Arquitectura de la capa de gestión de servicios.

2.1. Niveles del Sistema de gestión de servicios

2.1.1. Nivel Interfaz

Es el nivel de interacción con agentes y usuarios. A esta capa pertenecen cinco agentes del SCDIA: Agente Administrador de Agentes, Agente Gestor de Recursos, Agente Gestor de Aplicaciones, Agente de base de datos y Agente de Control de Comunicación. Los agentes del nivel interfaz hacen uso de los servicios del nivel medio.

El nivel interfaz es el encargado de establecer las pautas de la conversación entre los componentes del sistema y los agentes, como también, definir los esquemas de coordinación con los agentes de nivel superior.

2.1.2. Nivel Medio

Constituye el núcleo del sistema distribuido, provee servicios al nivel interfaz. Garantiza transparencia y seguridad en las transacciones, interoperabilidad de las aplicaciones y componentes de software, migración de agentes, objetos y/o recursos, comunicación ínter-proceso y provee un sistema de nombramiento para la localización de agentes y/o objetos.

1.1.3. Nivel de Acceso a Recursos

Esta capa está integrada por todos los manejadores asociados a recursos directos, tales como manejadores para acceso a datos relaciones, que implementan conexiones ODBC, JDBC, ADO, DAO, ADO.NET, manejadores de accesos a sistemas en tiempo real y a sistemas supervisores, acceso a sistema de planeación empresarial y gestión de recursos (ERP: Enterprise Resources Planning), acceso a sistemas de gestión de relaciones con los clientes (CRM), acceso a aplicaciones o componentes de software, y acceso directo a hardware específico.

Un esquema para acceso a recursos integra los componentes y recursos al sistema multiagente. El esquema se realiza implementado un traductor o proxy que comunica el recurso con los otros componentes o recursos del sistema. Para esto se pueden utilizar las tecnologías DCOM/COM (Distributed/Component Object Model) Java Beans, JNI (Java Native Interface) CORBA y XML (eXtended

Meta Language), que permiten acceder el recurso exportando su interfaz a un lenguaje de definición común.

Así, en este nivel se define un esquema para acceder y lograr la comunicación con todos los recursos ubicados en las diferentes plataformas computacionales, de una manera estándar y transparente.

CAPITULO III: Diseño del sistema de gestión de servicios

Siguiendo la metodología MAS-CommonKads extendida en este capítulo se diseñan los agentes pertenecientes al SGS. A continuación, se especifican los modelos de agentes, tareas, comunicación, coordinación e inteligencia. Estos modelos extraen las características principales del sistema a construir. Los modelos propuestos sirven como base para la implementación del SGS. Finalmente, se presenta la plataforma con la cuál se trabajó en el proceso de implementación.

1. Especificación de los agentes del Sistema de Gestión de Servicios

Para gestionar los servicios se especifican los siguientes tipos de agentes:

1.1. Nombre: Administrador de Agentes (AAA).

Tipo: Agente de Software.

Papel: Administrador del sistema multiagentes.

Descripción: se encarga de manejar, integrar y supervisar el estado del SCDIA. Este agente conoce la localización y estado de todos los agentes que existan en el sistema. El AAA dirige las migraciones de los agentes; cada agente que se mueva de un nodo a otro debe notificar al AAA el movimiento que ha efectuado; de manera que el agente administrador siempre tenga una vista ajustada al estado del sistema en tiempo real.

1.2. Nombre: Base de Datos (ABD).

Tipo: Agente de Software.

Papel: Gestor de Datos en el SCDIA.

Descripción: este agente se encarga de establecer el enlace con los lugares donde existan datos de interés para el proceso que se esté ejecutando, sea que estos datos provengan de bases de datos (relacionales, orientadas a objetos, tiempo real, etc.), de SCADAS, DCS, medidores, o cualquier otro dispositivo o aplicación que pueda almacenar datos. Además, el agente debe permitir el traslado de los datos entre los diferentes dispositivos y/o aplicaciones de una

manera transparente. Responde a peticiones, agentes de control, agentes actores y de medición, agentes coordinadores, agentes de aplicaciones, agentes de recursos, y especializados.

1.3. Nombre: Gestor de Aplicaciones (AGA).

Tipo: Agente de Software.

Papel: Administrador de aplicaciones del SCDIA.

Descripción: este agente se encarga de ubicar las aplicaciones que puedan ser requeridas por un proceso que se esté ejecutando, como por ejemplo, programas de cálculo numérico o simbólico, aplicaciones de inteligencia artificial, etc. Dichas aplicaciones pueden estar en cualquier servidor al que se tenga acceso y son requeridas por los agentes coordinadores, de bases de datos, especializados, de acceso a datos y administrador de recursos.

1.4. Nombre: Gestor de Recursos (AGR).

Tipo: Agente de Software

Papel: Administrador de recursos Hardware del SCDIA

Descripción: este agente se encarga de distribuir el uso de los dispositivos (hardware) necesarios en la ejecución de un proceso, como por ejemplo procesadores, dispositivos entrada/salida, dispositivos de almacenamiento, etc. Este agente puede ser accedido por cualquier agente del SCDIA. Los recursos pueden estar distribuidos y ser accedidos de forma remota.

1.5. Nombre: Control de Comunicación (ACC)

Tipo: Agente de Software.

Papel: Administrador de comunicaciones del SCDIA.

Descripción: es el encargado de mantener y controlar la comunicación entre sistemas multiagentes. Se encarga de traducir y manipular ontologías, y mantener un estado confiable del canal de comunicación.

2. Modelos para el diseño del Sistema de Gestión de Servicios

Para el diseño de los agentes se sigue los lineamientos y pautas de la metodología MAS-CommonKads extendida. A través del uso de los modelos de tareas, agentes, coordinación, inteligencia y comunicación se establecieron las características descriptivas del sistema multiagente. A continuación se exponen el conjunto de modelos.

2.1. Modelo de Agentes

2.1.1. Agente Administrador de Agentes (AAA)

Agente/Clase/Grupo

Nombre: Administrador de Agentes

Tipo: Agente de Software

Papel: Administrador del sistema multi-agentes

Posición: suministrador de información

Descripción: se encarga de manejar, integrar y supervisar el estado del sistema multiagente. Este agente conoce la localización y estado de todos los agentes que existan en el sistema. El AAA dirige las migraciones de los agentes; cada agente que se mueva de un nodo a otro debe notificar al AAA el movimiento que ha efectuado; de manera que el agente administrador siempre tenga una vista ajustada al estado del sistema en tiempo real

Objetivo: Administrar el SCDIA

Parámetros-Entrada: Información sobre los agentes del SCDIA: descripción, localización, estados.

Parámetros-Salida: Información sobre el estado del SCDIA

Condición de activación: Petición de un agente

Condición de éxito: Encontrar el agente en el diccionario de agentes y suministrar la información requerida.

Condición de fracaso: No encontrar el agente en el diccionario de agentes

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de SCDIA

Descripción: El AAA estará en capacidad de proveer información sobre cualquier agente del SCDIA

Servicios

Nombre: Descripción de Agentes

Parámetros-entrada: Nombre del Agente

Parámetros-salida: Descripción del agente: nombre, servicios, deseos, localización.

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de SCDIA

Nombre: Localización de Agentes

Parámetros-entrada: Nombre del Agente

Parámetros-salida: Ubicación actual del agente

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de SCDIA

Nombre: Estado de Agentes

Parámetros-entrada: Nombre del Agente

Parámetros-salida: Estado del agente

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de SCDIA

Nombre: Identificación de Agentes ejecutores de tareas

Parámetros-entrada: Nombre de la tarea

Parámetros-salida: Nombre y ubicación actual del (los) agente(s) que realiza(n) las tareas

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de SCDIA

Nombre: Migración de Agentes

Parámetros-entrada: Nombre del Agente, Ubicación de Destino

Parámetros-salida: Indicación de éxito o fracaso de la migración. Actualización del diccionario de agentes

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de SCDIA

2.1.2. Agente de Base de Datos (ABD)

Agente/Clase/Grupo

Nombre: Base de Datos

Tipo: Agente de Software

Papel: Gestor de Datos en el SCDIA

Posición: suministrador de información

Descripción: este agente se encarga de establecer el enlace con los lugares donde existan datos de interés para el proceso que se esté ejecutando, sea que estos datos provengan de bases de datos (relacionales, orientadas a objetos, tiempo real, etc.), de SCADAS, DCS, sensores, o cualquier otro dispositivo o aplicación que pueda almacenar datos. Además, el agente debe permitir el traslado de los datos entre los diferentes dispositivos y/o aplicaciones de una manera transparente. Responde a peticiones de agentes de bases de datos, agentes de control, agentes actuadores y sensores, agentes coordinadores, agentes localizadores y especializados

Objetivo: Obtención de Datos

Parámetros-Entrada: Solicitud de información por parte de un agente: Nombre del dato, tabla, registro al que se quiere acceder

Parámetros-Salida: Información solicitada

Condición de activación: Petición de un agente

Condición de finalización: El usuario no cuenta con los permisos necesarios para acceder a la información

Condición de éxito: Encontrar la información solicitada y entregársela al agente que la requirió

Condición de fracaso: No poder entregar la información solicitada al agente que la requirió

Lenguaje de Rep. del conocimiento: SQL, API´s, ACL

Ontología: Ontología de SCDIA

Descripción: El ABD estará en capacidad de obtener datos solicitados por cualquier agente del SCDIA

Servicios

Nombre: Búsqueda de información

Parámetros-entrada: Solicitud de información por parte de un agente: Nombre del dato, tabla, registro al que se quiere acceder

Parámetros-salida: Información solicitada

Lenguaje de Rep. del conocimiento: ACL, SQL

Ontología: Ontología de SCDIA

Nombre: Actualización

Parámetros-entrada: Información a ser colocada en la BD. Localización de la BD. Información sobre el usuario

Parámetros-salida: Notificación de actualización realizada

Lenguaje de Rep. del conocimiento: SQL, API´s, ACL

Ontología: Ontología de SCDIA

Nombre: Identificación de medios de transmisión de datos

Parámetros-entrada: Información solicitada, Ubicación, Tipo de Datos

Parámetros-salida: Medio a través del cual se hará la transmisión de información: ODBC, API´s, JDBC, Moscad, cadenas de bits, etc.

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de SCDIA

2.1.3. Agente Gestor de Aplicaciones (AGA)

Agente/Clase/Grupo

Nombre: Gestor de Aplicaciones

Tipo: Agente de Software

Papel: Administrador de aplicaciones del SCDIA

Descripción: este agente se encarga de ubicar las aplicaciones que puedan ser requeridas por un proceso que se esté ejecutando, como por ejemplo acceso a redes, programas de cálculo numérico o simbólico, aplicaciones de inteligencia artificial, envío y recepción de mensajes, etc. Dichas aplicaciones pueden estar en cualquier servidor al que se tenga acceso y son requeridas por los agentes

coordinadores, de bases de datos, localizadores, especializados, de acceso a datos y administrador de recursos

Objetivo: Ejecutar Aplicaciones

Parámetros-Entrada: Nombre de aplicación. Parámetros Necesarios para poder ejecutar la aplicación.

Parámetros-Salida: Ejecución de la aplicación que satisface la solicitud realizada

Condición de activación: Petición de ejecución de una aplicación por parte de un agente.

Condición de éxito: La aplicación es ejecutada

Condición de fracaso: No se puede ejecutar la aplicación

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología del SCDIA

Descripción: El agente AGA estará en la capacidad de ejecutar aplicaciones por solicitud de agentes del SCDIA

Objetivo: Localizar Aplicaciones

Parámetros-Entrada: Nombre de aplicación

Parámetros-Salida: Localización de la aplicación

Condición de activación: Petición de ejecución de una aplicación por parte de un agente. Petición de localización de una aplicación por parte de un agente del SCDIA

Condición de éxito: La aplicación es localizada

Condición de fracaso: No se puede localizar la aplicación

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología del SCDIA

Descripción: El agente AGA estará en la capacidad de localizar aplicaciones por solicitud de agentes del SCDIA

Servicios

Nombre: Ejecución

Parámetros-entrada: nombre de la aplicación a ejecutar

Parámetros-salida: Resultados de la ejecución de la aplicación

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de recursos del SCDIA

Nombre: Ubicación

Parámetros-entrada: nombre de la aplicación

Parámetros-salida: servidores de aplicaciones donde se puede ejecutar la aplicación

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de recursos del SCDIA

2.1.4. Agente Gestor de Recursos (AGR)

Agente/Clase/Grupo

Nombre: Gestor de Recursos

Tipo: Agente de Software

Papel: Administrador de recursos HW del SCDIA

Descripción: este agente se encarga de distribuir el uso de los dispositivos (hardware) necesarios en la ejecución de un proceso, como por ejemplo procesadores, dispositivos entrada/salida, dispositivos de almacenamiento, etc. Este agente puede ser accedido por cualquier agente de control, o por los dos agentes descritos anteriormente

Objetivo: Administrar Recursos

Parámetros-Entrada: Nombre del recurso. Tarea a realizar

Parámetros-Salida: Asignación de recursos

Condición de activación: Petición de uso de un recurso por parte de un agente

Condición de finalización:

Condición de éxito: Asignación de recursos

Condición de fracaso: No se puede asignar el recurso solicitado

Lenguaje de Rep. del conocimiento: Lenguaje Natural

Ontología: Ontología del SCDIA

Descripción: El agente AGR estará en la capacidad de administrar los recursos disponibles en el SCDIA

Servicios

Nombre: Ubicación

Parámetros-entrada: nombre del recurso solicitado

Parámetros-salida: ubicación del recurso solicitado

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de SCDIA

Nombre: Asignación

Parámetros-entrada: nombre del recurso. Localización

Parámetros-salida: asignación del recurso solicitado

Lenguaje de Rep. del conocimiento: ACL

Ontología: Ontología de SCDIA

2.1.5. Agente de Control de Comunicación (ACC)

Agente/Clase/Grupo

Nombre: Control de Comunicación

Tipo: Agente de Software

Papel: Administrador de comunicaciones del SCDIA

Descripción: es el encargado de mantener y controlar la comunicación entre sistemas multi-agentes. Se encarga de traducir, manipular ontologías, y mantener un estado confiable del canal de comunicación

Objetivo: Comunicar Sistemas Multiagentes

Parámetros-Entrada: Sistemas Multiagentes

Parámetros-Salida: Canal de Comunicación abierto

Condición de activación: Petición de inicialización de comunicación entre dos sistemas multiagentes

Condición de finalización: Fin de la comunicación

Condición de éxito: Canal de comunicación activo

Condición de fracaso: Problemas en el canal de comunicación

Lenguaje de Rep. del conocimiento: ACL, XML

Ontología: Ontología del SCDIA, otras ontologías

Servicios

Nombre: Inicialización

Tipo: Agente de software

Parámetros-entrada: Sistemas Multiagentes

Parámetros-salida: Canal de Comunicación activado

Lenguaje de Rep. del conocimiento: ACL,XML

Ontología: Ontología de SCDIA, otras ontologías

Nombre: Envío de Mensaje

Tipo: Agente de software

Parámetros-entrada: Agente remitente, mensaje, Agente, receptor

Parámetros-salida: Mensaje de Respuesta

Lenguaje de Rep. del conocimiento: ACL, XML

Ontología: Ontología de SCDIA, otras ontologías

Nombre: Fin de la comunicación

Tipo: Agente de software

Parámetros-entrada: Ninguno

Parámetros-salida: Finalización del mensaje

Lenguaje de Rep. del conocimiento: Lenguaje Natural

Ontología: Ontología de SCDIA, otras ontologías

2.2. Modelo de Comunicación

En el modelo de comunicación se especifican los casos de usos para cada uno de los agentes del SGS. Estos casos de usos que sirven de base para construir los diagramas de secuencias de las comunicaciones que se llevan a cabo entre agentes.

2.2.1. Casos De Uso Agente Gestor De Aplicaciones (AGA)

Tiene una tabla interna que le permite buscar los agentes disponibles para tareas específicas. Se este agente se encarga de ubicar las aplicaciones que puedan ser requeridas por un proceso que se esté ejecutando, como por ejemplo, acceso a redes, programas de cálculo numérico o simbólico, aplicaciones de

inteligencia artificial, envío y recepción de mensajes, etc. Dichas aplicaciones pueden estar en cualquier servidor al que se tenga acceso y son requeridas por los agentes del SCDIA etc.

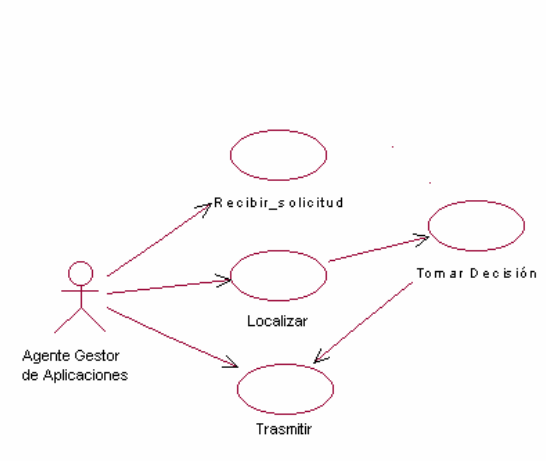


Figura 9. Diagrama de Casos de Uso del AGA

Nombre: Recibir Solicitud

Resumen: El AGA recibe y procesa la solicitud de los A. de SCDIA

Actores:

Agentes de SCDIA $\Rightarrow$ AGA

Precondiciones: Debe existir comunicación entre el AGA y los demás A. del SCDIA

Excepción: $\neg$ Precondición

Postcondición: Recepción de la orden

Nombre: Localizar la aplicación

Resumen: El AGA debe localizar los lugares (servidores de aplicaciones) donde se encuentren las aplicaciones requeridas, si no la encuentra pregunta al ACC para buscar la aplicación en otro SMA.

Actores: AGA, AGA $\Rightarrow$ ACC

Precondiciones: Debe existir comunicación entre el AGA y los servidores de aplicaciones

Excepción: $\neg$ Precondición

Postcondición: Servidor(es) de aplicaciones o agente especilazado localizado(s)

Nombre: Tomar Decisión

Resumen: El AGA realiza una toma de decisión para dar respuesta al agente que solicita la aplicación. Para dar respuesta a la solicitud se tiene que tomar en cuenta aspectos como balanceo de carga, nodo más próximo, etc.

Actores: AGA ⇒ A. SCDIA

Precondiciones: Solicitud de tarea recibida

Postcondicion: Decisión Tomada

Nombre: Trasmitir

Resumen: El AGA tramite la información recolectada sobre la aplicación solicitada.

Actores: A. Gestor de Aplicaciones ⇒ A. SCDIA

Precondiciones: Debe existir comunicación entre el AGA y los agentes del SCDIA.

Postcondición: Información recibida.

Figura 10. Diagrama de Estados del AGA

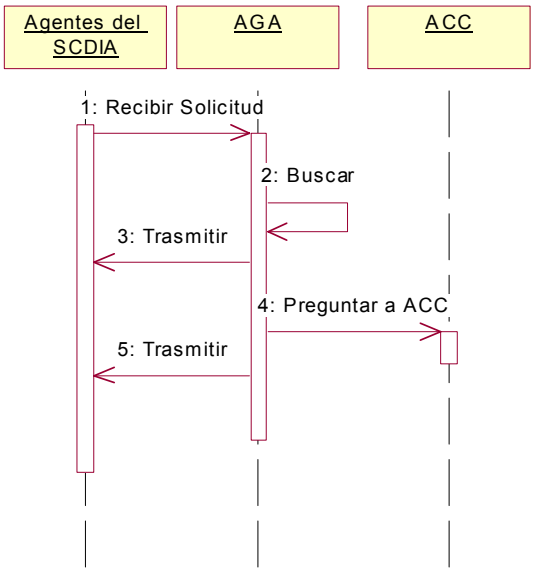


Figura 11. Diagrama de Interacción del AGA

2.2.2. Casos de uso Agente Gestor de Recursos (AGR)

Los agentes gestores de recursos (AGR) son parte del middleware de los SCDIA. Ellos se encargan de distribuir el uso de los dispositivos (hardware) necesarios en la ejecución de un proceso, como por ejemplo procesadores, dispositivos de entrada/salida, dispositivos: de almacenamiento, etc. Este agente puede ser accedido por cualquier agente del SCDIA.

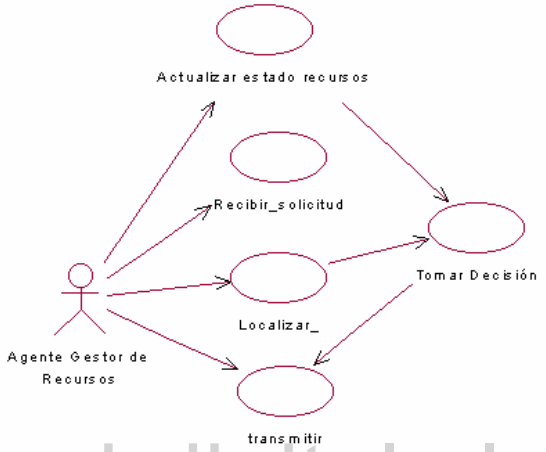


Figura 12. Diagrama de Casos de Uso del AGR

Nombre: Actualizar estado de recursos

Resumen: El AGR actualiza el estado del recurso.

Actores: AGR, agentes atados a recursos

Precondiciones: acceso al recurso.

Excepción: ¬ Precondición

Postcondición: Recurso accedido

Nombre: Recibir Solicitud

Resumen: El AGR está en espera de peticiones de uso de recursos por parte de los A. de SCDIA

Actores: Agentes de SCDIA ⇒A. Gestor de Recursos

Precondiciones: Debe existir comunicación entre el AGR y los recursos a asignar.

Excepción: ¬ Precondición

Postcondición: Recepción de la orden

Nombre: Localizar Recurso

Resumen: El AGR busca en la tabla de recursos local el recurso solicitado, si no lo encuentra pregunta al ACC si se encuentra el recurso en otro SMA.

Actores: Agentes de SCDIA ⇒AGR, AGR⇒ ACC

Precondiciones: Debe existir comunicación entre el AGR y los demás A. del SCDIA

Excepción: ¬ Precondición

Postcondición: Resultado de la búsqueda: Nombre, estado y localización del recurso.

Nombre: Tomar decisión

Resumen: El AGR toma decisión sobre el uso y la asignación de los recursos.

Actores: A. Gestor de Recursos

Precondiciones: Recurso disponible, tiempo para gestionar.

Excepción: ¬ Precondición

Postcondición: Recurso utilizado de forma óptima.

Nombre: Trasmitir

Resumen: El AGR tramite la información recolectada sobre el o los recursos solicitados (estado y localización).

Actores: AGR ⇒ Agentes del SCDIA

Precondiciones: Debe existir comunicación entre el AGR y el Agente que solicita el recurso.

Excepción: ¬ Precondición

Postcondición: Información recibida.

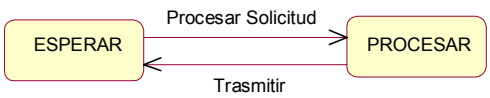


Figura 13. Diagrama de Estado Agente Gestor de Recursos

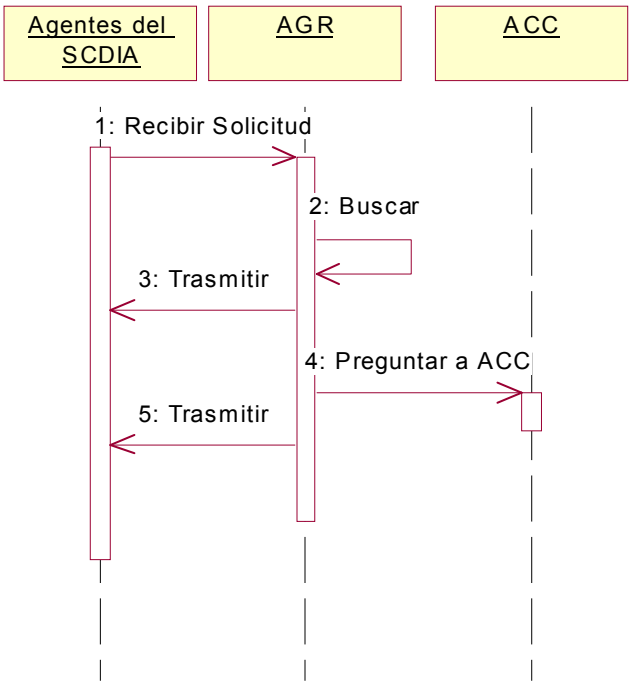


Figura 14. Diagrama de Interacción del AGR

2.2.3. Casos De Uso Agente Base De Datos (ABD)

Este agente se encarga de establecer el enlace entre los lugares donde existan datos de interés para el proceso que se esté ejecutando, sea que estos datos provengan de bases de datos (relacionales, orientadas a objetos, tiempo real, etc.), de SCADAS, DCS, sensores, o cualquier otro dispositivo o aplicación que pueda almacenar datos. Este agente también realiza la gestión de los medios de almacenamiento de los datos. Además, el agente permite el traslado de los datos entre los diferentes dispositivos y/o aplicaciones de una manera transparente. Responde a peticiones de los agentes del SCDIA.

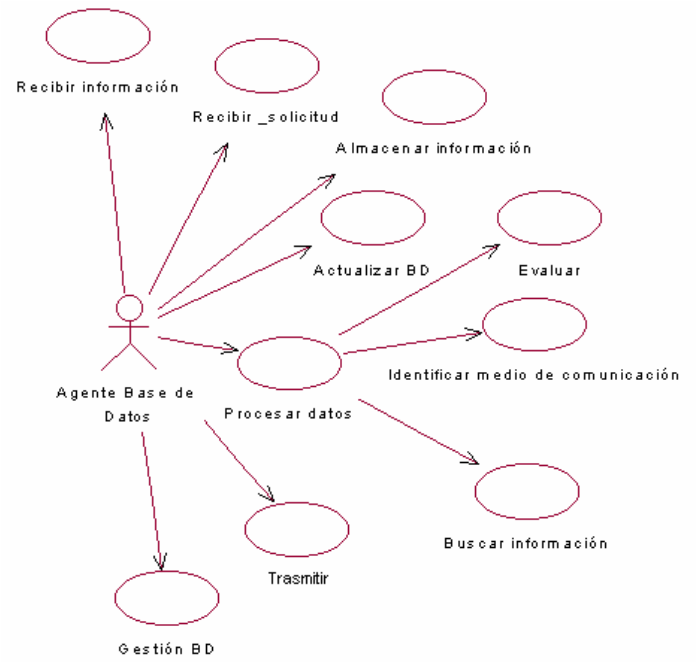


Figura 15. Diagrama de Casos de Uso del Agente BD

Nombre: Recibir Información

Resumen: El agente BD recibe información de los agentes del sistema

Actores: Agentes del SCDIA ⇒ A. BD

Precondiciones: Existencia de comunicación y de información a recibir

Excepcion: ¬Precondición

Postcondicion: Información recibida

Nombre: Recibir Solicitud

Resumen: El agente BD recibe una solicitud de información de algún agente del sistema

Actores: A. SMA ⇒ A. Base de Datos

Precondiciones: Existencia de la solicitud y la comunicación

Excepcion: ¬Precondición

Postcondicion: Solicitud recibida

Nombre: Almacenar (Guardar)

Resumen: La información se anexa a la información existente en la BD

Actores: A. SCDIA ⇒A. Base de Datos

Precondiciones: Existencia de la información a almacenar

Excepcion: ¬Precondición. Memoria llena

Postcondicion: Información almacenada

Nombre: **Actualizar**

Resumen: La información recibida se rescribe en la dirección asociada

Actores: A. Base de Datos

Precondiciones: Existencia de la información a actualizar y orden de actualizar

Excepcion: ¬Precondición

Postcondicion: Actualización de la información en la BD

Nombre: **Procesar solicitud**

Resumen: El agente BD determina las características las solicitudes recibidas (permisología, medio de comunicación, etc.)

Actores: A. Base de Datos

Precondiciones: Existencia de la solicitud y métodos de toma de decisiones

Excepcion: ¬Precondición

Postcondicion: Decisión tomada

Nombre: **Evaluar**

Resumen: Toda solicitud a la BD conlleva un análisis de los permisos, para control de acceso y manejo de prioridades

Actores: A. Base de Datos

Precondiciones: Solicitud Recibida

Excepcion: ¬Precondición

Postcondicion: Solicitud aprobada o denegada

Nombre: **Identificar medio de comunicación**

Resumen: El A. BD debe identificar que clase de protocolo de transmisión de datos es necesario utilizar para obtener los datos requeridos

Actores: Agentes de SCDIA $\Rightarrow$ A. BD

Precondiciones: El A. dispone de los protocolos de comunicación que se requieren para buscar los datos.

Excepción: $\neg$ Precondición

Postcondición: Protocolo elegido

Nombre: Hacer Petición (buscar información)

Resumen: El A de BD debe hacer la petición de los datos al medio que los almacena (BD, SCADA, Sensor, etc.)

Actores: Agentes de SCDIA ⇒ A. BD

Precondiciones: El A. dispone de los permisos necesarios para acceder a los datos.

Excepción: ¬ Precondición

Postcondición: Datos Requeridos

***Nombre:* Entregar datos (transmitir)**

Resumen: Una vez recibidos los datos el A. BD debe entregar los datos al agente que hizo la petición.

Actores: A. BD ⇒ Agentes de SCDIA

Precondiciones: Debe existir comunicación entre el A. BD y los demás A. del SCDIA

Excepción: ¬ Precondición

Postcondición: Entrega de datos requeridos

***Nombre:* Gestión de BD**

Resumen: El A. BD realiza tareas de mantenimiento de la base de dato en función de las especificaciones de diseño (respaldos, etc.)

Actores: A. Base de Datos

Precondiciones: Tareas especificadas de los manejadores de BD.

Postcondicion: Gestión realizada

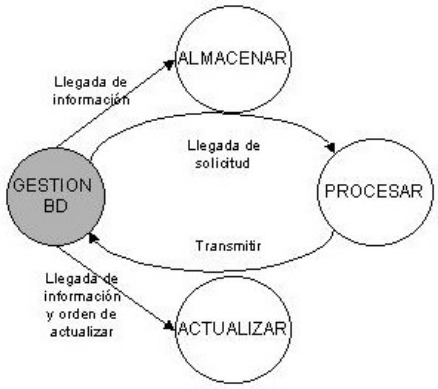


Figura 16. Diagrama de Estados del Agente BD

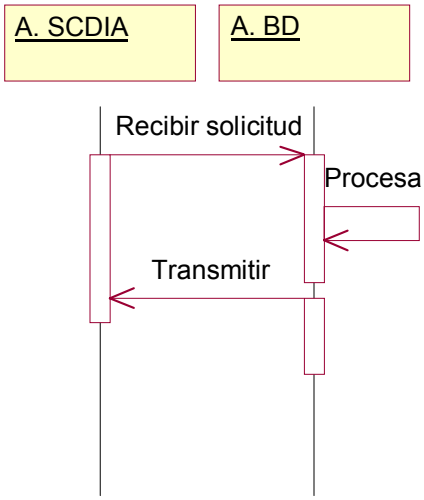


Figura 17. Diagrama de Interacciones del Agente Base de Datos

2.2.4. Casos De Uso Agente Administrador De Agentes (AAA)

EL AAA es parte fundamental de los estándares como FIPA, y se incluye en el middleware debido a dos funciones principales que debe desarrollar: La localización de Agentes y la Migración de Agentes. A continuación se presenta un caso de estudio para cada uno de ellos.

Localización de Agentes

El AAA se encarga de manejar, integrar y supervisar el estado del SMA. Este agente conoce la localización y estado de todos los agentes que existan en el sistema. Cada agente que se mueva de un nodo a otro debe notificar al AAA el movimiento que ha efectuado; de manera que el agente administrador siempre tenga una vista ajustada al estado del sistema en tiempo real.

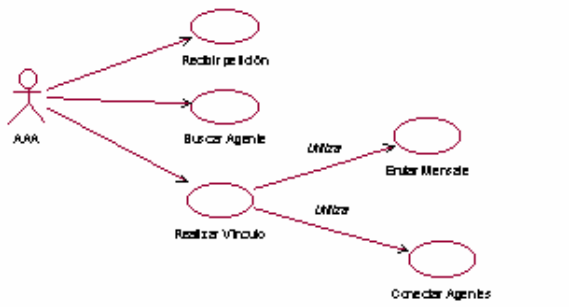


Figura 18. Caso de Uso del AAA para localizar agentes

Caso de Uso del AAA para localizar agentes

Nombre: **Recibir petición**

*Resumen:*EL A. Administrador recibe una petición de buscar un agente en específico para poder establecer un vínculo con él.

Actores: Agentes del SCDIA $\Rightarrow$ AAA

Precondiciones: Petición aceptada

Excepcion: Falla de la comunicación

Postcondicion: Petición procesada

Nombre: **Buscar Agente**

Resumen: el A. Administrador debe consultar en la tabla de localización de agentes y obtener el nodo donde se encuentra hospedado el agente buscado

Actores: AAA, Agentes del SMA.

Precondiciones: Exista el agente a buscar

Excepcion: No se consigue el agente

Postcondicion: Agente localizado

Nombre: **Realizar Vínculo**

Resumen: Comunicar agente localizado y agente que realiza petición

*Actores:*AAA $\Rightarrow$ A. Localizado

Precondiciones: Agente localizado

Excepcion: No se consigue el agente

Postcondicion: Agente localizado

Nombre: **Enviar Mensaje**

Resumen: Comunicar a A. Localizado petición del AGA o AGR.

*Actores:*A. Administrador $\Rightarrow$A. Localizado

Precondiciones: Agente localizado

Excepcion: Falla de Comunicación

Postcondicion: Mensaje transmitido y aceptado

NOMBRE: **Conectar Agentes**

Resumen: enviar mensaje de la localización del agente buscado al agente que realiza la petición con la finalidad de realizar el vínculo

Actores: AAA ⇒ Agente del SCDIA

Precondiciones: Agente localizado

Excepcion: Falla de Comunicación

Postcondicion: Vínculo realizado

Caso de Uso del AAA para Migración de agentes

Existen varias técnicas y modos de migración para agentes y objetos. Para lograr migración de agentes es necesario contar con una plataforma comunicacional con ciertas características, que permita utilizar el protocolo TCP/IP y el protocolo HTTP. En el caso de los sistemas multi-agentes se proveen canales que permiten que los agentes viajen a través de los diferentes nodos de la red. A continuación se modelan los casos de uso para la migración de agentes en el SCDIA.

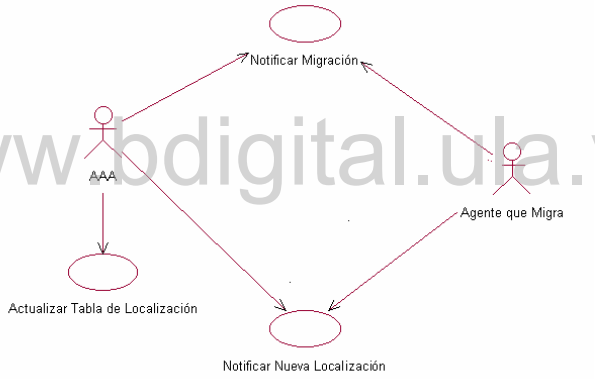


Figura 19. Casos de Uso del AAA para migrar agentes

Nombre: **Notificar Migración**

Resumen: El agente que migra notifica al AAA que va a Migrar

Actores: A. que Migra => AAA

Precondiciones: Agente a migrar está autorizado. DESCRIPCIÓN:

Excepcion: Falla de Comunicación

Postcondicion: Migración aceptada

Nombre: **Notificar nueva localización**

Resumen: Se notifica al AAA que la trasmisión ha sido un éxito y el agente ha sido reiniciado en el nodo destino

Actores: A. que Migra => AAA

Precondiciones: Nodo destino activo

Excepcion: Falla de la comunicación

Postcondicion: Tabla de localización actualizada

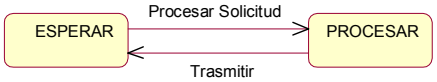


Figura 20. Diagrama de Estado Administrador de Agentes

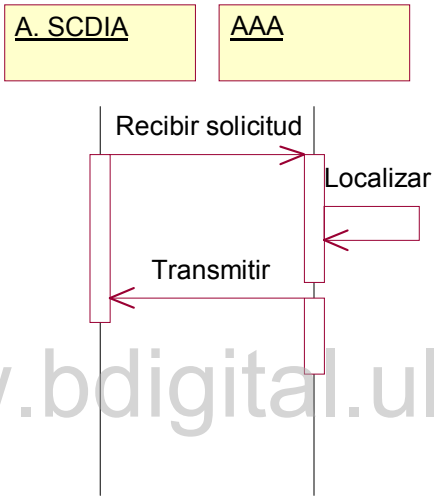


Figura 21. Diagrama de Interacciones del Agente Administrador de Agentes

2.2.5. Casos De Uso Agente De Control De Comunicación (ACC)

El ACC es el encargado de mantener y controlar la comunicación entre SMA. Se encarga de traducir, manipular ontologías, y mantener un estado confiable del canal de comunicación. Para el ACC se modelan dos diagramas de casos de uso; el primer caso es para solicitar a otro SMA un agente, recurso o aplicación; el segundo de los casos mostrando las búsquedas en el SMA local

Caso de Uso Preguntar a otro SMA

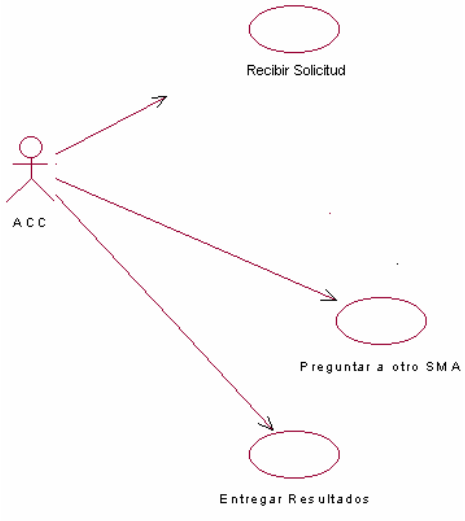


Figura 22. 1er Caso ACC. Preguntar a otro SMA sobre un agente, recurso o aplicación

Nombre: **Recibir Solicitud**

Resumen: Agentes del SCDIA pregunta al ACC sobre un agente, aplicación o recurso.

Actores: AAA ⇒ ACC, AGA ⇒ ACC, AGR ⇒ ACC

Precondiciones: Nombre del agente, aplicación o recurso en formato válido

Excepcion: No existe conexión con otros SMA

Postcondicion: Petición aceptada

Nombre: **Preguntar a otro SMA**

Resumen: ACC pregunta a otro ACC en otro SMA sobre la localización de un componente o aplicación, recurso o agente.

Actores: ACC ⇒ ACC otro SMA

Precondiciones: Existe conexión con otros SMA.

Excepcion: No existe conexión con otros SMA

Postcondicion: Petición aceptada

Nombre: **Entregar resultados**

Resumen: ACC envía los resultados al Agente del SCDIA que realizó la solicitud.

Actores: ACC ⇒ AAA, ACC ⇒ AGA, ACC ⇒ AGR

Precondiciones: Existe comunicación.

Excepclón: No existe comunicación

Postcondicion: Resultados recibidos

Caso de Uso Búsqueda en el SMA local

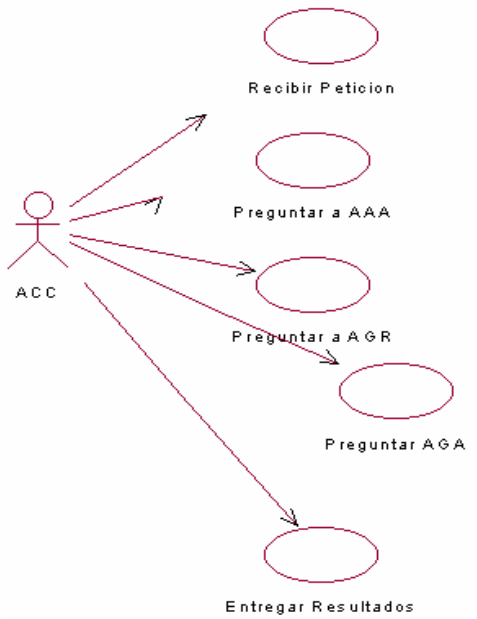


Figura 23. Caso de Uso Búsqueda en el SMA local

Nombre: **Recibir Petición**

Resumen: Un ACC de otro SMA realiza la petición de búsqueda de un agente, aplicación o recurso.

Actores: ACC $\Rightarrow$ ACC

Precondiciones: Existe comunicación con el agente del SMA solicitante.

Excepcion: No existe comunicación con el agente del SMA solicitante.

Postcondicion: Petición Aceptada.

Nombre: **Preguntar al AAA**

Resumen: ACC pregunta al AAA sobre la localización de un agente y su estado.

Actores: ACC $\Rightarrow$ AAA

Precondiciones: Nombre del agente a buscar válido.

Excepcion: No existe comunicación

Postcondicion: Respuesta del AAA.

Nombre: **Preguntar al AGR**

Resumen: ACC pregunta al AGR sobre la localización de un recurso y su estado.

Actores: ACC ⇒ AGR

Precondiciones: Nombre del recurso a buscar válido.

Excepcion: No existe comunicación

Postcondicion: Respuesta del AGR.

Nombre: **Preguntar al AGA**

Resumen: ACC pregunta al AGA sobre la localización de un componente o aplicación.

Actores: ACC ⇒ AGA

Precondiciones: Nombre de la aplicación o componente a buscar válido.

Excepcion: No existe comunicación

Postcondicion: Respuesta del AGA.

Nombre: **Entregar resultados**

Resumen: ACC envía los resultados al ACC del otro SMA.

Actores: ACC ⇒ ACC.

Precondiciones: Existe comunicación.

Excepcion: No existe comunicación

Postcondicion: Resultados recibidos.

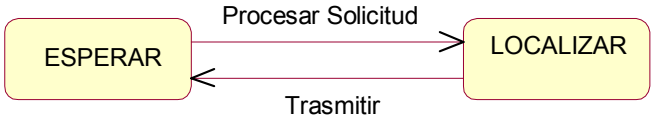


Figura 24. Diagrama de Estado Agente Controlador de Comunicación

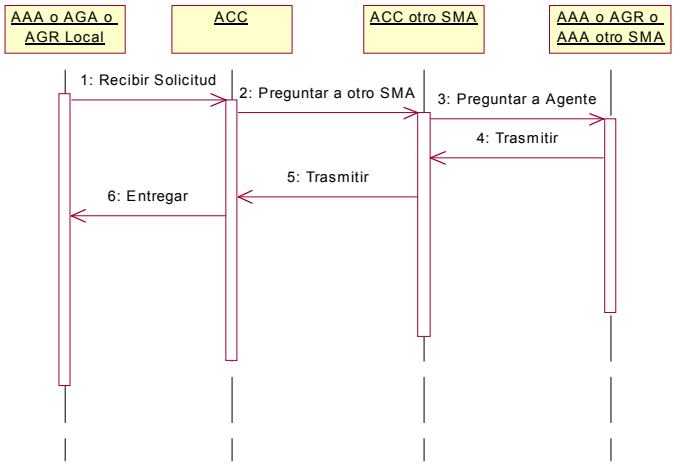


Figura 25. Diagrama de Interacciones del Agente de Control de Comunicación

2.3. Modelo de tareas

El modelo está integrado al SCDIA, es decir, también incluye las características asociadas con los procesos de control. Se han extraído las tareas asociadas con los agentes del SGS.

El modelo de tareas permite describir las actividades relacionadas para alcanzar un objetivo; en nuestro caso, un objetivo de control. El desarrollo del modelo de tareas tiene por fin documentar la situación actual y futura del SCDIA, facilitar la gestión de cambios, y ayudar a estudiar el alcance y viabilidad del sistema inteligente. Las tareas cognitivas que se deseen implementar se detallarán en el modelo de la experiencia, mientras que las tareas de comunicación se detallarán en un modelo de comunicación (comunicación humana) o coordinación (comunicación con agentes).

2.3.1. Tareas y Servicios del SCDIA

Del Modelo de Casos de Uso se desprenden el siguiente conjunto de tareas que se desglosan en función de las entidades descritas anteriormente.

Tareas de manejo de información
Almacenar información –1
Actualizar información – 2
Procesar (del A. BD) – 3
Evaluar (del A. BD) – 4
Buscar información – 5
Gestión BD – 6
Tareas de localización
Localizar recurso
Localizar agente especializado de recurso
Localizar aplicación
Loc. Ag. Especializado en aplicación
Asignación de tarea/recurso
Balanceo de carga u otro criterio de optimización
Tareas de activación de agentes
Inicializar agente – 1
Asociar nombre, nodo y estado – 1.1
Comunicar agente de recurso – 1.2
Migrar agente – 2
Cambiar estado – 3
Administrar seguridad – 4
Destruir agente – 5

Tabla 2. Tareas del SCDIA

Procesamiento de datos
Búsqueda de información
Almacenamiento de datos
Actualización

Identificación de medios de transmisión de datos.
Seguridad de datos
Evaluación de situaciones
Coordinación de actividades (metas)
Planificación de tareas
Descripción de agentes
Localización de agentes
Actualización de agentes
Migración de agentes
Seguridad a nivel agentes
Capacidad de generar restricciones
Ubicación de aplicaciones
Inicialización
Finalización de comunicación
Envío de mensajes.

Tabla 2. Tareas del sistema de gestión de servicios

El modelo de tareas permite mostrar la descomposición funcional del sistema multiagente.

2.3.1.1. Tareas de manejo de información

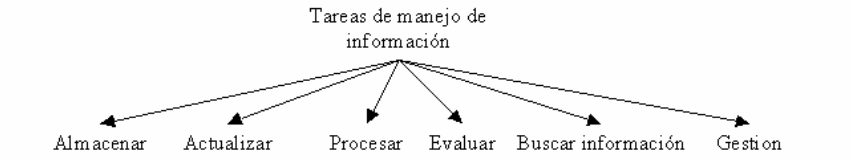


Figura 26. Árbol de descomposición de tareas de manejo de información

Tarea: Almacenar información

Objetivo: Los datos provenientes de cualquiera de los agentes del SMA se anexan a los que ya existen.

Rendimiento: Los datos deben almacenarse correctamente, ya que de allí provienen los históricos del sistema

Descripción: El almacenamiento de la información permite tener históricos de las diferentes variables del proceso o del SMA

Precondición: Tener un nuevo valor a almacenar y la identificación de la variable
Estructura de Control: El almacenamiento de las variables es una tarea común en las Bases de datos que no requiere subdividirse
Tiempo de Ejecución: Debe ser instantáneo
Frecuencia: Frecuencia Relativa a la llegada de la variable
Tipo de Descomposición: N/A
Ingrediente:
Entrada: Variable o Dato a almacenar
Salida: Variable o dato almacenado
Contenido: La variable o dato que llega a la base de datos se almacena en una localidad de memoria

Tarea: Actualizar información
Objetivo: Los datos provenientes de cualquiera de los agentes del SMA se sobrescriben en la dirección asociada
Rendimiento: La actualización se debe realizar en la dirección correspondiente, con eso se garantiza un rendimiento eficiente

Descripción: Algunos datos no requieren valores históricos, sino solo el valor actual, así este se actualiza siempre en la misma dirección
Precondición: Tener un nuevo valor a actualizar y la orden de actualizar el dato
Estructura de Control: La actualización es una tarea común en BD que no requiere de divisiones
Tiempo de Ejecución: Debe ser instantáneo
Frecuencia: Frecuencia Relativa a la llegada de la variable
Tipo de Descomposición: NA
Ingrediente:
Entrada: Variable o Dato a actualizar
Salida: Variable o dato actualizado
Contenidos: La variable o dato que llega a la base de datos se actualiza en una localidad de memoria

Tarea: **Procesar agente base de datos**

Objetivo: Las solicitudes se procesan para poder ejecutarse

Rendimiento:

Descripción: El agente BD procesa las solicitudes recibidas en cuanto al control de acceso y la prioridad de la solicitud

Precondición: Tener una solicitud de información

Estructura de Control: El procesamiento requiere de una etapa de evaluación y una de búsqueda de información

Tiempo de Ejecución: Debe ser instantáneo

Frecuencia: Frecuencia Relativa a la llegada de la solicitud

Tipo de Descomposición: El procesamiento de las solicitudes necesita de un proceso de evaluación para ver si la solicitud procede, de ser así se prosigue con la búsqueda de la información, de lo contrario se envía un mensaje de falla

Ingrediente:

Entrada: Solicitud de búsqueda de información

Salida: Información solicitada o mensaje de falla

Contenidos: La información solicitada se envía al agente solicitante, de lo contrario se envía un mensaje de falla

Entorno: Sistema Multiagente

Normas: Siempre se debe devolver un valor aunque sea de falla

Restricciones: Si no se consigue algún dato que debería de estar en la BD, se debe enviar un mensaje de falla además de una alarma al agente humano

Sistemas: SMA

Tarea: **Evaluar agente Base de datos**

Objetivo: Las solicitudes se evalúan para ver si proceden

Descripción: Toda solicitud a la BD conlleva un análisis de los permisos, para control de acceso y manejo de prioridades

Precondición: Tener una solicitud de información

Estructura de Control: La evaluación de solicitudes no requiere de subtareas

Tiempo de Ejecución: Debe ser instantáneo

Frecuencia: Frecuencia Relativa a la llegada de la solicitud

Tipo de Descomposición: NA

Ingrediente:

Entrada: Solicitud de búsqueda de información

Salida: Aprobación o no de la solicitud

Contenidos: La solicitud se evalúa para ver si procede.

***Tarea:* Buscar Información agente Base de datos**

Objetivo: Las solicitudes aprobadas se ejecutan

Rendimiento:

Descripción: El agente BD busca información solicitada

Precondición: Tener una solicitud de información aprobada

Estructura de Control: La búsqueda de información no requiere de subtareas

Tiempo de Ejecución: Debe ser instantáneo

Frecuencia: Frecuencia Relativa a la llegada de la solicitud

Tipo de Descomposición: NA

Ingrediente:

Entrada: Solicitud de búsqueda de información aceptada

Salida: Información encontrada o no

Contenidos: Si se consigue la información se transmite y si no se envía una falla o una señal de alarma

***Tarea:* Gestión Base de datos**

Objetivo: Tareas de rutinas en la base de datos

Rendimiento:

Descripción: El A. BD realiza tareas de mantenimiento de la base de dato en función de las especificaciones de diseño (respaldos, etc.)

Precondición: Estar especificado en el diseño y que se cumpla el tiempo para realizarse

Estructura de Control: Puede involucrar otras subtareas

Tiempo de Ejecución: Debe ser instantáneo

Frecuencia: Frecuencia Relativa a las especificaciones de diseño

Tipo de Descomposición: NA

2.3.1.2. Tareas de localización

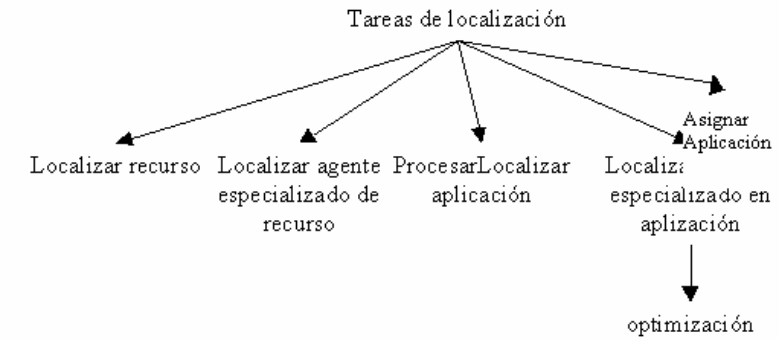


Figura 27. Árbol de descomposición de tareas de localización

***Tarea:* Localizar Recurso**

Objetivo: Encontrar el recurso y conectarlo al agente que lo solicita.

Descripción: Cuando se solicita un recurso se busca en la tabla de recursos y se provee la conexión entre agentes.

Rendimiento: Bueno

Pre-Condición: Existe Agente que provee el recurso. Comunicación Posible.

Tiempo Ejecución: Corto-Medio

Frecuencia: Muy Frecuentemente.

***Tarea:* Localizar Agente Especializado de Recursos**

Objetivo: Encontrar Agente especializado y conectarlo al agente que lo solicita.

Descripción: Cuando se solicita un recurso se busca en la tabla de recursos y se provee la conexión entre agentes.

Rendimiento: Bueno

Pre-Condición: Existe Agente que provee el recurso. Comunicación Posible.

Tiempo Ejecución: Corto-Medio

Frecuencia: Muy Frecuentemente.

Ingrediente:

Contenido: Identificador del recurso a localizar

Capacidad: Medio de búsqueda

*Descripción:*El agente gestor de recursos tiene que tener las herramientas y algoritmos apropiados de búsqueda para encontrar al agente solicitado.

Entorno: Ambiente Seguro

Normas: No afectar ejecución de otros agentes o procesos de otros sistemas.

***Tarea:* Localizar Aplicación**

Objetivo: Encontrar la aplicación y conectarlo al agente que lo solicita.

Descripción: Cuando se solicita una aplicación se busca en la tabla de servidores de aplicaciones y se provee la conexión entre agente solicitante y la aplicación.

Rendimiento: Bueno

Pre-Condición: Existe Agente especializado. Comunicación Posible.

Tiempo Ejecución: Corto-Medio

Frecuencia: Muy Frecuentemente.

***Tarea:* Localizar Agente Especializado en Aplicación**

Objetivo: Encontrar el agente especializado de aplicación y conectarlo al agente que lo solicita.

Descripción: Cuando se solicita un agente de aplicación se busca el agente especializado que provee la aplicación y se provee la conexión entre agente solicitante y el agente especializado.

Rendimiento: Bueno

Pre-Condición: Existe Agente especializado. Comunicación Posible.

Tiempo Ejecución: Corto-Medio

Frecuencia: Muy Frecuentemente.

***Tarea:* Balanceo de Carga**

Objetivo: Distribuir carga de procesamiento entre servidores .

Descripción: Este agente puede ejecutar algoritmos de balanceo de carga de procesamiento y distribuir procesamiento entre varios agentes.

Rendimiento: Bueno

Pre-Condición: Existe Agente que provee el recurso. Comunicación Posible.

Tiempo Ejecución: Corto

Frecuencia: Regular.

***Tarea:* Asignar Aplicación**

Objetivo: Asignar al agente solicitante la aplicación o aplicaciones que requiere.

Descripción: Este agente puede asignar a un agente del SCDIA una aplicación o informar sobre la capacidad de otro agente de prestar el servicio solicitado.

Rendimiento: Bueno

Pre-Condición: Existe Agente que provee la aplicación. Comunicación Posible.

Tiempo Ejecución: Corto

Frecuencia: Regular.

2.3.1.3. Tareas de activación de agentes

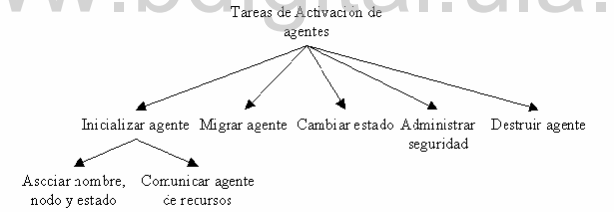


Figura 28. Árbol de tareas de activación de agentes

***Tarea:* Inicializar Agente**

Objetivo: Colocar en ejecución en el nodo local o remoto el agente requerido.

Descripción: Coloca en funcionamiento al agente requerido, le otorga un nombre y participa al agente gestor de recursos y al agente gestor de aplicaciones las funciones que el agente ofrece.

Rendimiento: Bueno

Pre-Condición: Buscar agente de la biblioteca de agentes.

Estructura-Control: Asociar Nombre, Nodo y Estado. Comunicar a agentes de recursos

Tiempo Ejecución: Lineal

Frecuencia: Muy Frecuentemente.

Ingrediente:

Nombre: Agente Inicializado

Contenido: La salida de la tarea es el agente que se ha creado en un espeacio de memoria primaria.

Capacidad: Memoria Disponible

Descripción: Suficiente memoria primaria y/o secundaria para crear el agente

Entorno: Ambiente Seguro

Normas: No afectar ejecución de otros agentes o procesos de otros sistemas.

Restricciones: Ejecutarse en el espacio de memoria local.

***Tarea:* Asociar Nombre, Nodo y Estado**

Objetivo: Asignar al agente solicitado un nombre unívoco referido a su localidad y función.

Descripción: Coloca en la tabla de nombres el nuevo agente inicializado el estado activo y la localidad, con la finalidad de los demás agentes puedan comunicarse con este agente.

Rendimiento: Bueno

Pre-Condición: Existe Agente

Tiempo Ejecución: Lineal

Frecuencia: Muy Frecuentemente.

Ingrediente:

Nombre: Nombre a asignar

Contenido: El nombre a asignar debe ser unívoco.

Nombre: Capacidad

Descripción: El nombre no debe estar siendo usado por otro agente, recurso o aplicación.

Entorno: Ambiente Seguro

Normas: No afectar ejecución de otros agentes o procesos de otros sistemas.

Restricciones: Ejecutarse en el espacio de memoria local.

***Tarea:* Comunicar a agentes de recursos**

Objetivo: Actualizar las tablas de servicios administrada por los agentes de recursos.

Descripción: Los agentes de recursos (aplicaciones, datos y hardware) deben conocer las funciones que ofrece el nuevo agente inicializado.

Rendimiento: Bueno

Pre-Condición: Existe Agente. Asociado Nombre, Nodo y Estado.

Tiempo Ejecución: Lineal.

Frecuencia: Muy Frecuentemente.

Ingrediente:

Nombre: Lista de Tareas

Contenido: Es una lista que contiene todas las tareas que puede realizar el agente que ha sido inicializado

Capacidad: Tareas realizables

Descripción: Todas las tareas incluidas en la lista deben ser realizables por el agente, tomando en cuenta todos los aspectos de seguridad, integridad, confiabilidad y disponibilidad de recursos y aplicaciones.

Entorno: Ambiente Seguro

Normas: No afectar ejecución de otros agentes o procesos de otros sistemas.

Restricciones: Ejecutarse en el espacio de memoria local.

***Tarea:* Migrar Agente**

Objetivo: Migrar Agente de un nodo a otro.

Descripción: El agente Administrador de Agentes (AAA) es el encargado de administrar las migraciones de agentes, y tener actualizada la tabla de localización correspondiente. El AAA debe conocer la localización de los agentes que componen el sistema multi-agente.

Rendimiento: Bueno.

Pre-Condición: Nodos Origen y Destino Activos. Existe Agente. Asociado. Comunicación Posible.

Tiempo Ejecución: Lineal.

Frecuencia: Frecuentemente.

Ingrediente:

Nombre: Nodos origen y destino

Contenido: El nodo origen y destino son los nodos de red entre los que el agente va a moverse

Capacidad: Canal de comunicación abierta.

Descripción: El canal de comunicación debe estar disponible.

Ingrediente:

Nombre: Memoria disponible en nodo destino.

Descripción: En el nodo destino debe estar disponible la suficiente memoria para alojar el agente que va a migrar.

Entorno: Ambiente Seguro

Normas: No afectar ejecución de otros agentes o procesos de otros sistemas.

Restricciones: Ejecutarse en el espacio de memoria local.

***Tarea:* Cambiar Estado Agente**

Objetivo: Cambiar el estado de funcionamiento del Agente Inactivo/Activo.

Descripción: El agente Administrador puede controlar el estado de ejecución de los agentes que forman parte del sistema multi-agente, puede activar o desativar a un agente que se encuentre en el nodo local o en un nodo remoto.

Rendimiento: Bueno

Pre-Condición: Existe Agente. Comunicación Posible.

Tiempo Ejecución: Lineal.

Frecuencia: No Muy Frecuentemente.

Ingrediente:

Nombre: Agente

Contenido: Agente que se le aplica el cambio de estado

Capacidad: Agente puede cambiar de estado

Descripción: El agente al cual se le aplica el cambio de estado debe estar en capacidad de aceptar el cambio de estado, ya que es posible que el agente este realizando una tarea prioritaria que no puede ser interrumpida.

Entorno: Ambiente Seguro

Normas: No afectar ejecución de otros agentes o procesos de otros sistemas.

Restricciones: Ejecutarse en el espacio de memoria local.

Tarea: **Destruir Agente**

Objetivo: Borrar de memoria el agente solicitado

Descripción: El agente Administrador puede borrar de memoria primaria y secundaria el agente solicitado.

Rendimiento: Bueno.

Pre-Condición: Existe Agente. Comunicación Posible.

Tiempo Ejecución: Corto.

Frecuencia: No Muy Frecuentemente.

Ingrediente:

Nombre: Agente

Contenido: Agente al que se le aplica la eliminación

Capacidad: Agente puede cambiar de estado

Descripción: El agente al cual se le aplica el cambio de estado debe estar en capacidad de aceptar el cambio de estado, ya que es posible que el agente este realizando una tarea prioritaria que no puede ser interrumpida.

Entorno: Ambiente Seguro

Normas: No afectar ejecución de otros agentes o procesos de otros sistemas.

Restricciones: Ejecutarse en el espacio de memoria local.

***Tarea:* Administrar Seguridad**

Nombre: Administrar Seguridad.

Objetivo: Mantener la confiabilidad e integridad de los recursos y agentes del sistema multi-agente.

Descripción: Administra los permisos y recursos sobre agentes, acredita o desacredita migraciones, mantiene tabla de permisos, cambia, agrega o elimina permisos.

Pre-Condición: Existe Agente. Comunicación Posible.

Tiempo Ejecución: Corto

Frecuencia: Muy Frecuentemente.

Ingrediente: Agente

Contenido: Agente a acreditar o desacreditar

Capacidad: Agente puede cambiar de estado

Descripción: El agente al cual se le aplica el cambio de estado debe estar en capacidad de aceptar el cambio de estado, ya que es posible que el agente este realizando una tarea prioritaria que no puede ser interrumpida.

Entorno: Ambiente Seguro

Normas: No afectar ejecución de otros agentes o procesos de otros sistemas.

Restricciones: Ejecutarse en el espacio de memoria local.

2.4. Modelo de Coordinación

Este modelo describe las políticas y protocolos de coordinación entre agentes, que tienen que darse para lograr los objetivos y tareas propuestas. A continuación se sigue este modelo para describir estos protocolos para el SGS.

2.4.1. Esquemas de coordinación

Esquema de Coordinación

***Nombre:* Solicitud**

Objetivo: obtener la acción solicitada

Tipo: Predefinida

Estrategia: intervenciones

Planificación:

Tipo: distribuida

Técnica: pase de mensajes, estructura de objetos

Mecanismo de comunicación directa:

Librería: protocolo bajo TCP/IP

Metalenguaje:

Nombre: Agent Comunication Language (ACL)

Medio: protocolo bajo TCP/IP

Ontología:

Nombre: de Recursos

Representación: Objetos

Agentes que la conocen: agentes del SCDIA

Descripción: ésta ontología contiene los predicados, acciones, y conceptos para el manejo de los recursos y aplicaciones del SGS.

Esquema de Coordinación

***Nombre:* de contrato**

Objetivo: contratar y realizar un conjunto de servicios

Tipo: Predefinida

Estrategia: intervenciones

Ontología: de recursos

Esquema de Coordinación

***Nombre:* Pregunta**

Objetivo: obtener una información sobre un recurso o aplicación

Tipo: Predefinido

Estrategia: intervenciones

Ontología: de recursos

Esquema de Coordinación

***Nombre:* Subasta**

Objetivo: Realizar el servicio obteniendo los mejores recursos del ambiente

Tipo: Predefinido

Estrategia: intervenciones

Ontología: de recursos

Esquema de Coordinación

***Nombre:* Corredor**

Objetivo: Seleccionar agentes para realizar un servicio

Tipo: Predefinido

Estrategia: intervenciones

Ontología: de recursos

2.5. Modelo de inteligencia

El modelo de inteligencia corresponde con una arquitectura que intenta emular el razonamiento humano. Se cuenta con estructuras interrelacionadas que procesan información del sistema y producen decisiones. En la próxima parte se presentan los atributos básicos que componen este modelo para los agentes del SGS.

Mecanismo de razonamiento

Fuente de Información: Base de datos, Base de reglas, variables del ambiente.

Fuente de activación: Intervención

Tipo de inferencia: deductiva

Estrategia de razonamiento: sistema de reglas

Mecanismo de aprendizaje

Nombre: Redes neuronales de retropropagación, Árboles de decisión

Tipo: supervisado

Fuente de Aprendizaje: históricos de datos

Mecanismo de actualización: Cambio de valores en las reglas

Experiencia

Representación: reglas de un sistema experto

Grado de confiabilidad: moderada

Motor de inferencia: motor de reglas

Conocimiento estratégico

Valor del conocimiento: alto

Agente que lo produce: Gestor de aplicaciones, gestor de recursos, gestor de base de datos otros agentes del SMA.

Proceso que lo generó: Consultas a base de datos, consulta a sistemas de relaciones con los clientes (CRM).

Grado de confiabilidad: alto

3. Descripción de la capa Intermedia

3.1. Nombramiento

Un sistema multiagente (SMA) debe contar con un administrador de nombres, que debe tener como mínima cualidad la adjudicación de nombres únicos asociados a los agentes, esta cualidad es llamada unicidad. La unicidad permite la comunicación coherente entre los agentes, ya que permite identificar correctamente al remitente y destinatario de cada mensaje, logrando una comunicación efectiva.

Para el SGS se dispone de un administrador de nombres que cumple con la unicidad de nombres, además permite obtener la localización del agente. Los nombres se describen por una dirección dividida en varias partes: el nombre, el nodo, el puerto y el nombre de la plataforma de mensajes, dispuestas de la siguiente forma:

nombre_local@nodo:puerto/plataforma

La primera parte es el nombre local del agente, que puede ser una palabra de hasta 255 caracteres alfanuméricos, y es usada para identificarlo en un contexto local. Esta parte del nombre puede ser designada por el usuario o por

otro agente. Los únicos nombres locales designados por el SGS son los agentes propios del sistema, tales como el Administrador (AAA), el Gestor de Aplicaciones (AAA), el Gestor de Recursos (AGR), el agente de base de datos (ABD) y el agente de Control de Comunicación (ACC).

La segunda parte del nombre corresponde al nombre del nodo (*host*) de la red, que generalmente es el nombre de red asignado al computador o servidor donde se encuentra el agente. Seguido de los dos puntos debe estar el número de puerto TCP por donde se realiza la comunicación, que generalmente es el número 1099.

Por último se asigna la plataforma donde opera el agente nombrado, que en nuestro caso será JADE/SCDIA.

Los nombres no son fijos y varían según la localización del agente, por ejemplo; cuando se realiza una acción de migración de un agente de un nodo a otro de la red, la parte nodo del nombre cambia y todas las otras partes permanecen iguales, si ocurre el caso de que existe un agente con el mismo nombre local en el nodo destino se niega la migración.

El administrador de nombres garantiza nombres únicos y globales referenciados por un nombre local y un nombre de localización, logrando un compromiso entre libertad de nombramiento y asignación directa.

3.2. Interoperabilidad

La heterogeneidad de sistemas de hardware y software en las organizaciones industriales y empresariales, hace que los procesos de integración sean largos y complejos. Esta heterogeneidad se traduce en un número grande de interfaces que deben comunicarse entre sí, teniendo que acoplar objetos, procedimientos y tipos de datos.

Para dar solución al problema de integración, se dispone del modelo de componente, que permite envolver cualquier recurso, datos o aplicación bajo una misma interfaz. Los dos tecnologías más difundidas son la COM (Component Object Model, desarrollada por Microsoft®) y la llamada JavaBeans (desarrollada

por Sun Microsystems®). La primera es más extendida y es usada en aplicaciones Windows; la segunda es para sistemas basados en la máquina virtual de Java.

Estas dos tecnologías están integradas al SGS, permitiendo operar con distintas plataformas de software y hardware. Debido a que el SCDIA está desarrollado en Java, es posible instalar los contenedores de agentes a lo largo de varios nodos con distintos sistemas operativos y con diferentes tipos de hardware.

La interacción con COM, realizada a través de un *proxy*[3] desarrollado con JNI (Java Native Interface), permite integrar bibliotecas de recursos o aplicaciones completas al SMA. También es posible interactuar con protocolos de red tales como el HTTP, utilizado para la Web, el SMTP utilizado para envío de correo electrónico, o el XML, que se perfila como un estándar para integración con aplicaciones para la internet y aplicaciones legadas.

Toda esta capacidad que posee la capa el SGS para interactuar con otros sistemas ya desarrollados es una cualidad prioritaria, ya que le permite a cada agente medir y actuar sobre el ambiente, configurado por la heterogeneidad de aplicaciones y hardware que dispone la organización.

3.3. Transparencia

En sistemas complejos y distribuidos la arquitectura de software generalmente está organizada en capas, que le otorgan la cualidad de transparencia a las operaciones.

En la arquitectura del SGS la organización por capas hace que las operaciones de la capa inferior sean transparentes a las capas superiores, simplificando los procesos y la ejecución de funciones.

La arquitectura de tres capas divide las funciones del medio según el elemento que manejan, en la capa de más abajo el elemento principal es el componente, en la segunda el servicio y en la más alta el agente.

Según estos tres tipos de abstracción, en cada capa cada elemento se muestra como un objeto del tipo correspondiente, ocultando los detalles involucrados con las subtareas.

---

[3] Aplicación intermedia que comunica dos o más módulos de software.

3.4. Mensajería y comunicación interprocesos

En sistemas multiagentes la mensajería juega un papel muy importante, ya que es la base de la comunicación entre agentes. Una de las diferencias más notorias entre objetos y agentes es la cualidad de comunicación asíncrona. Cuando se programa con objetos los mensajes son tradicionalmente síncronos. Se sigue un hilo de ejecución y este se detiene en espera de la respuesta del mensaje:

```
Servidor servidor(dirección);
servidor.iniciarConexión();          <-------  Mensaje
Libros = servidor.preguntar(autor);  <-------  Mensaje
If (Libros.Longitud > 0 )
Imprimir(Libros);
        else
            Imprimir("No hay libros publicados para este autor");
```


El código anterior pregunta en un servidor remoto sobre la existencia de libros para un autor determinado, y luego los muestra al usuario. Se envían dos mensajes sincronos desde el objeto "cliente" al objeto "servidor". El hilo de ejecución se detiene hasta obtener la respuesta de cada mensaje. En este tipo de mensajería solo existe un contrato simple: se pide un servicio y se obtiene. En el caso de ocurrir error se notifica a través de un código o de una excepción.

Cuando se habla de programación multiagente el tipo de mensajería es diferente: en primer lugar el agente cuando envía un mensaje no espera por la respuesta para seguir ejecutando sus tareas, existe un protocolo independiente basado en performativas y ontologías que procesa los mensajes y decide que hacer en el momento que estos llegan.

Además cuando se envía un mensaje la respuesta esperada no está predefinida, no tiene un formato específico, es decir, la información no es procesada a través de tipos de datos o parámetros, sino a través de un lenguaje

para agentes que admite descripciones sobre elementos. En conclusión, no existen llamadas a procedimientos sino actos de habla entre agentes. Esta característica hace de la programación basada en agentes una herramienta poderosa en la solución de problemas complejos y dinámicos, ya que permite una interacción más flexible entre los módulos que forman parte de la aplicación, y por ende, de la organización.

El SCDIA utiliza un sistema de mensajes para dar apoyo al lenguaje ACL (Agent Comunication Language). El sistema de mensajes está basado en la serialización de objetos, esta herramienta permite la grabación y recuperación de objetos en un canal (archivo, protocolo de red, pantalla, etc.). Los objetos serializados guardan el estado de sus atributos. Estos tipos de objetos deben implementar la interfaz Serializable de la jerarquía de clases de Java.

4. Capa de acceso a recursos

Ya descritas los modelos y entornos que se utilizan en este trabajo para el acceso a recursos y para el desarrollo del SMA, en esta sección se explicará la arquitectura para integrar los modelos basados en componentes con el entorno multiagente.

El SGS está escrito en Java, por lo tanto, la plataforma puede estar distribuida y sus programas ser ejecutados en diferentes sistemas operativos sin tener que realizar diferentes compilaciones; estableciendo una única interfaz para acceso a recursos. Por ejemplo, para acceder al sistema archivo, se cuenta con un único conjunto de clases destinado a gestionar las operaciones sobre él, sin importar que se esté trabajando con implementaciones diferentes con distinta interfaz propietaria. Esto es posible porque el lenguaje está basado en el concepto de máquina virtual, que un procesador de software para transformar el código intermedio generado en el proceso de compilación a instrucciones de máquina que hacen las mismas operaciones en los distintos sistemas operativos.

Una de las desventajas que presenta este concepto, es que no es posible acceder de manera completa a todos los recursos del sistema operativo donde se ejecuta el programa, ya que existen una gran cantidad de características

particulares, interfaces, hardware, y recursos que no pueden ser accedidos vía máquina virtual.

Las opciones disponibles para el acceso a recursos son dos: una de ellas es realizar un proxy o intermediario utilizando CORBA, que es un estándar para objetos distribuidos, o utilizar JNI (Java Native Interface), una característica que brinda el lenguaje para escribir código en un segundo lenguaje nativo, que pueda acceder directamente al sistema operativo.

Para la capa de acceso a recursos se propone el desarrollo de un marco basado en componentes. Los componentes son módulos de software reutilizables, persistentes, y disponibles para varios lenguajes y aplicaciones. Como se nombró en el primer capítulo, existen dos estándares bastantes extendidos, el primero de ellos es para sistemas Windows y se denomina COM (Component Object Model), el otro es específicamente para la plataforma Java, y es conocido como JavaBeans, que aunque no son totalmente componentes, poseen muchas de sus características. Los *JavaBean* pueden ser de dos tipos, del tipo sesión (*session*) y del tipo entidad (*entity*); los del tipo sesión conceptualmente deben encapsular la lógica de negocio, por su parte los del tipo entidad deben encapsular el acceso a datos. Los *JavaBean* forman parte estructural del entorno J2EE (Java 2 Enterprise Edition), un entorno muy extendido en el campo industrial y empresarial.

Los componentes permiten encapsular funciones, objetos, variables, y servicios, para poder ser utilizados con una única interfaz. Debido a las cualidades de los componentes, en la actualidad la mayoría de las aplicaciones y lenguajes, brindan componentes propios para permitir acceso a sus servicios. También la mayoría de los lenguajes permiten construir fácilmente nuevos componentes accesibles a otras aplicaciones, como también herramientas con interfaces amigables para el desarrollo de componentes.

Por todo lo dicho anteriormente, los componentes se constituyen como una herramienta eficaz en el logro de acceso a recursos ya sean de hardware, software, distribuidos o centralizados. El problema se plantea entonces, en términos de realizar el vínculo con los conceptos de los SMAs. Los SMAs manejan ontologías para lograr comunicación, ejecutar acciones y entender conceptos y

relaciones, y los componentes derivados de la programación orientada a objetos, se expresan en métodos y propiedades. La capa de acceso a recursos debe resolver este vínculo, haciendo que los conceptos y acciones que manejan las ontologías tengan relación con los componentes instalados en cada nodo de la red.

Se propone utilizar el elemento denominado "*MatchComponent*"(ver figura 30) que realiza el enlace entre los dos conceptos: ontología y componente. Este elemento realiza un enlace entre los atributos de los conceptos y las propiedades de los componentes, entre las acciones de los agentes y los métodos de cada componente.

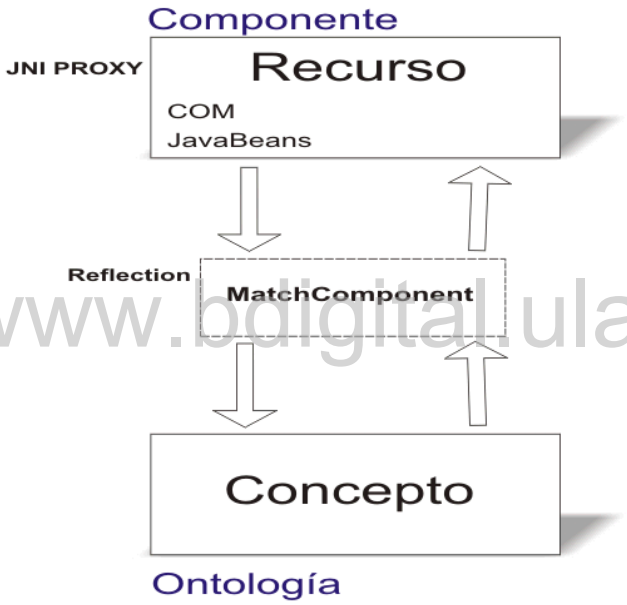


Figura 29. Modelo de la capa de recursos

Para lograr este acometido, se utiliza una de las características de los lenguajes de alto nivel orientados a objetos, por lo tanto, también del lenguaje Java, que se denomina reflexión (*reflection*). La reflexión es el concepto de los programas que trabajan sobre otros programas, de las clases que trabajan sobre otras clases, es decir, una biblioteca que provee un conjunto de servicios de invocación, depuración, inspección, de clases y métodos.

La reflexión permite inspeccionar totalmente los métodos y atributos de los objetos en tiempo de ejecución, conocer sus parámetros y tipos de datos. Esta

cualidad permite que para el modelo propuesto, se logre inspeccionar los elementos de la ontología de recursos y relacionarlos dinámicamente con los elementos de los componentes.

En el cuadro de recursos (ver figura 29), si bien se admite dos tipos de componentes, la conexión con componentes tipo COM no es directa, debido a que las bibliotecas de clases de Java no cubren este tipo de tecnología, Para dar respuesta a este problema se desarrolló un *proxy* entre Java y el componente COM. Este *proxy* se desarrolló utilizando JNI (Java Native Interface)[20], que permite vinculación directa con programas de compilación nativa. Este tipo de modelo puede ser extendido para otras plataformas no Windows, ya que JNI también está disponible también para otros sistemas.

El modelo basado en interfaz nativa, permite una conexión directa y eficiente con COM y no agrega otro sistema de software o plataforma al modelo. La desventaja de la JNI, es que la portabilidad de los programas compilados se pierde si se utilizan diferentes plataformas, pero la vinculación reflexiva con los elementos de la ontología, hace que no sea necesario mantener esta portabilidad ya que los elementos de la ontología vinculados con los componentes pueden trasmitirse de un nodo a otro de la red.

CAPITULO IV: Implementación del Sistema de Gestión de Servicios

En este capítulo se explica el proceso de implementación del SGS, se presentan las herramientas utilizadas y se muestra el diseño y técnicas para la construcción del sistema. En la implementación es necesario incluir características tales como heterogeneidad, transparencia y comunicación, así como también, un acceso a los recursos de forma uniforme.

JADE es un entorno de desarrollo para Sistemas Multiagentes escrito en Java (ver apéndice B). El SGS utiliza los servicios de esta plataforma. Se escogió este entorno por varias razones; está totalmente implementada en Java, cumple con las especificaciones FIPA, y permite comunicación bajo el modelo RMI y IIOP-CORBA. Fue desarrollado por el TILAB (Telecom Italia Laboratory) y la Universidad de Parma, Italia. Se distribuye bajo licencia LGPL (Lesser General Public License)[4], por lo tanto es posible disponer bajo ciertas restricciones de los programas fuentes y ejecutables. JADE es un conjunto de bibliotecas de clases, que ofrecen servicios y funcionalidades para la construcción y administración de sistemas multiagentes. Entre las capacidades de la plataforma está la característica de interoperabilidad, heredada de Java, que permite acceder a plataformas heterogéneas. También ofrece clases para la gestión de ontologías y la gestión de comunicación, sistema de nombramiento, administración de agentes, utilitarios, y otras características importantes inherentes a los sistemas multiagentes.

2. Arquitectura del Sistema Programado de Gestión de Servicios

En primer lugar, se diseñó una jerarquía de clases que implementa el diseño del SGS (ver figura 30). Esta jerarquía tiene como clase principal a *Agent*. De esta clase derivan los tipos de agentes: administrador de agentes (*AdministratorAgent*), gestor de recursos (*ResourcesAgent*), gestor de

[4] Esta tipo de licencia para código abierto puede accederse en el sitio web http://www.gnu.org

aplicaciones (*AppAgent*), gestor de base de datos (*BDAgent*), y del agente de control de comunicación (*CCAgent*). Cada agente está asociado con uno o más comportamientos (*Behaviour*). Las clases que derivan de *Behaviour* proveen un ciclo de acciones para el agente con el cual se encuentran asociadas. Un agente puede tener varios comportamientos que son ejecutados cuando se cumplen sus condiciones de inicialización, que generalmente son intervenciones o condiciones particulares del entorno. Los agentes que poseen interfaz gráfica para interactuar con el usuario, deben ser subclases de la clase *GUIAgent*. Por otra parte, las ontologías que se quieran incluir, deben ser definidas como extensión de la clase *Ontology*.

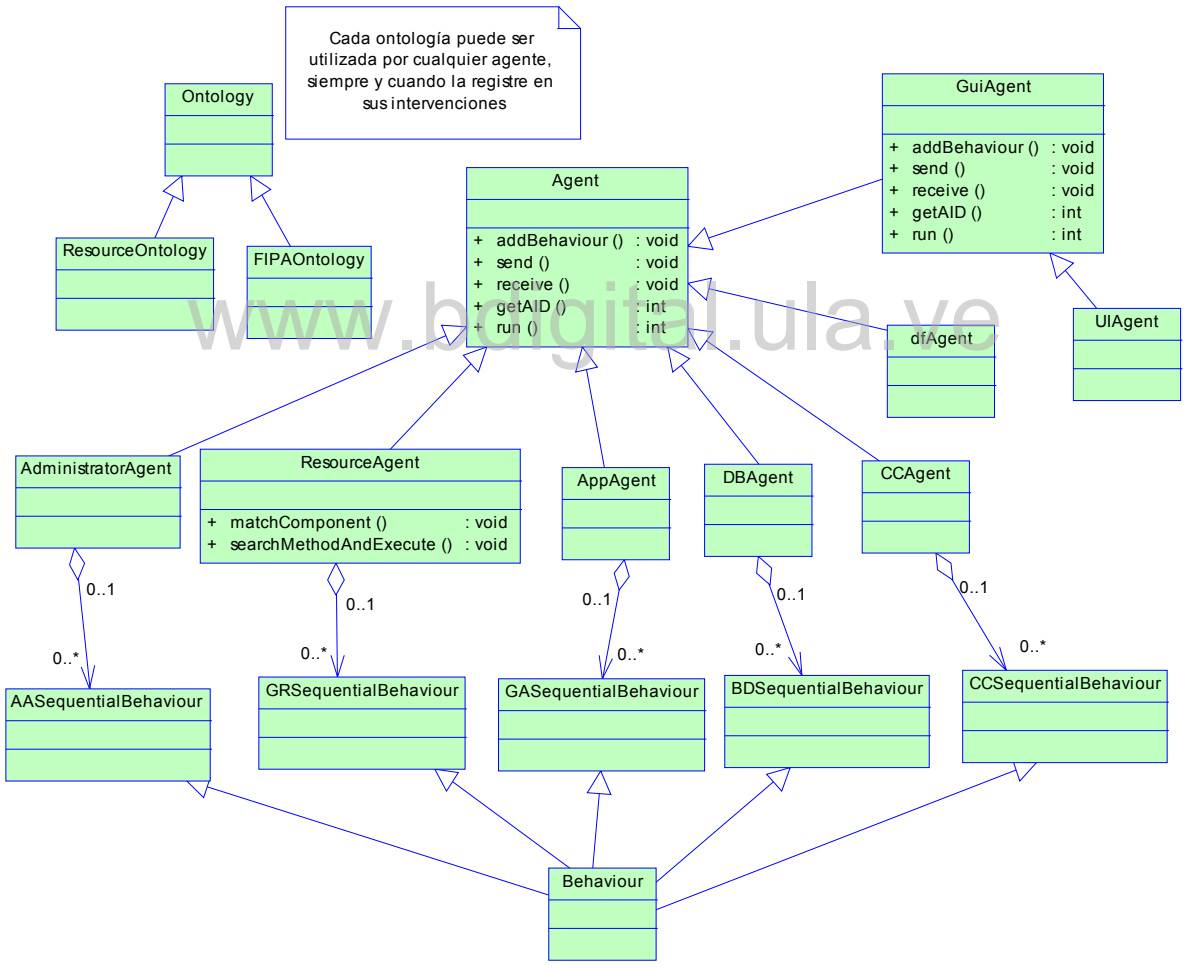


Figura 30. Jerarquía de clases para el SGS

La jerarquía de clases está diseñada en función de cumplir con los parámetros de diseño explicados en el capítulo tres. Basada en esta jerarquía se

provee de las herramientas básicas para la administración de agentes, tales como inicialización, suspensión, destrucción, clonación, copia y migración de agentes. Además posee el servicio de introspección, que permite conocer las conversaciones que se llevan cabo entre los agentes y monitorear remotamente las plataformas.

Como se expone en el diseño conceptual del SGS, existe un agente administrador de agentes (AAA) que coordina todas las operaciones básicas que se sucedan en el entorno. Para cada plataforma es necesario contar con un agente de este tipo, por lo tanto cada agente cuenta con un administrador del que depende, y con el cuál debe conversar para realizar cualquier servicio administrativo.

Siguiendo la teoría de agentes, toda interfaz de usuario que sirve para interactuar con el entorno de agentes es vista como un agente, tiene un espacio y nombre en la lista de agentes y es subclase de *GuiAgent*. El otro agente que no forma parte del diseño del SGS es el llamado facilitador (*directory facilitator:df*), que permite la búsqueda de agentes y servicios en el entorno, y que es especificado por FIPA[5].

Los demás agentes en la lista corresponden a los del diseño conceptual: agente administrador de recursos (AGR), agente administrador de aplicaciones (AGA), agente de base de datos (ABD), y el agente de control de comunicaciones (ACC).

Los agentes poseen estado, que corresponde con el hecho de que estén realizando una acción (activo) o estén suspendidos esperando por el reinicio de sus operaciones (suspendido). También puede estar creados en memoria pero no activos (inicializado), o pueden estar borrados de la memoria principal (destruido). Hay otros estados que corresponden con el hecho de que se estén realizando operaciones sobre ellos, como clonación (clonado) o migración (moviéndose).

Se dispone de una aplicación para la administración del entorno multiagente (ver figura 31). Ella permite visualizar el estado, configuración, nombre, dirección y demás características de los agentes. También es posible llevar cabo labores de administración a través de ella. Esta aplicación de

administración, que forma parte del entorno JADE, se modificó y extendió para cumplir con el diseño del SGS.

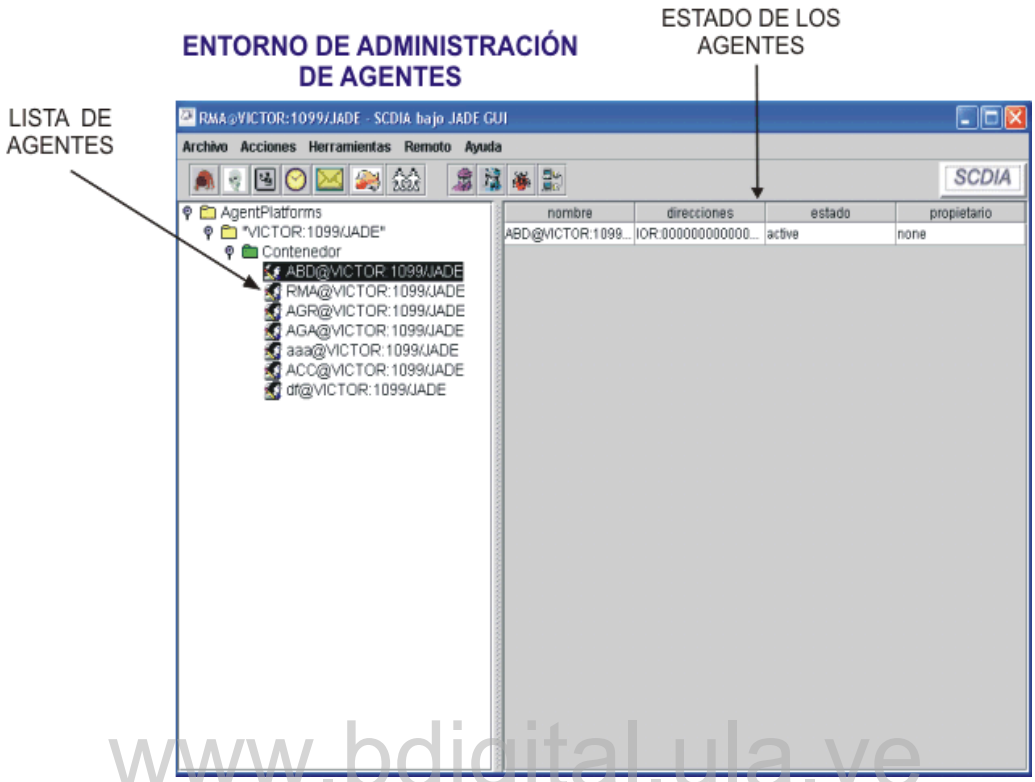


Figura 31. Entorno para administración de Agentes

3. Arquitectura de los agentes

Para realizar la implementación de cada agente, se parte de una arquitectura común. Esta arquitectura saca provecho de las últimas herramientas provistas por la programación orientada a objetos, tales como hilos de ejecución (threads), reflexión, serialización, e invocación remota de métodos.

Siguiendo el modelo de agentes planteado en el capítulo anterior, se programaron el conjunto de agentes del SGS. Para cada agente se estableció un comportamiento principal, que admite otros tipos de comportamientos, tales como los basados en máquinas de estados y los de tipo reactivo. Los comportamientos permiten el accionar de los protocolos de coordinación entre agentes. Entre estos protocolos se incluyen los más extendidos y especificados por FIPA, tales como el *contract-net*, *query* y *auction*.

JADE provee una jerarquía de clases para implementar los comportamientos y los protocolos de coordinación. En el caso de los comportamientos se eligió una estructura que permite la inclusión de sub-protocolos de forma dinámica, es decir, en consecuencia del estado y modelo de inteligencia del agente en un determinado momento éste reaccionará ejecutando un determinado tipo de comportamiento que corresponda al estado del entorno que se presente.

En el caso de los protocolos de coordinación, se siguen las directrices planteadas por el modelo de coordinación. Se tienen un conjunto de protocolos para coordinar tareas con el objetivo de alcanzar determinada meta; este conjunto puede ser extendido derivando una clase que procese de manera automática las intervenciones específicas para el protocolo. Por ejemplo, para el caso del *contract-net* (ver figura 32), se dispone de una clase que procesa las intervenciones de iniciación, propuestas y contratación automáticamente, solo es necesario especificar las acciones que se ejecutan en el desarrollo de la conversación. Los comportamientos proveen autonomía y los protocolos de coordinación proveen esquemas de colaboración y de competencia en comunidades de agentes.

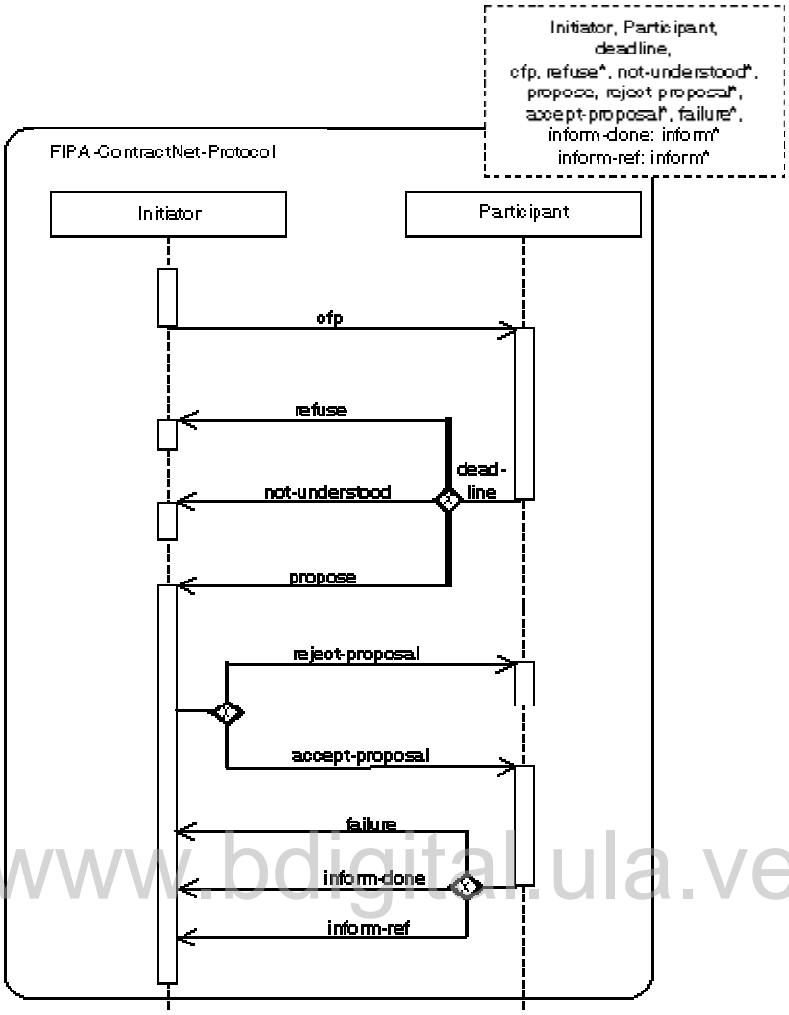


Figura 32. Modelo de interacción del Contract-net[500]

También es posible agregar varios comportamientos a un determinado agente. Por ejemplo, a todos los agentes del SGS se les agregó un comportamiento para responder a las preguntas del agente gestor de recursos (ver figura 33). Cuando el remitente es el AGR y se utilice el protocolo *QUERY*[5] se ejecuta el comportamiento *ResourceQueriesBehaviour* que incluye el manejo de las intervenciones para este protocolo. Este procedimiento de inserción de varios comportamientos se realiza también para los otros agentes del SGS, de manera que estos agentes posean múltiples comportamientos que se inician en función de las intervenciones que se sucedan.

[5] El protocolo QUERY está definido por la FIPA en sus especificaciones. ver http://www.fipa.org

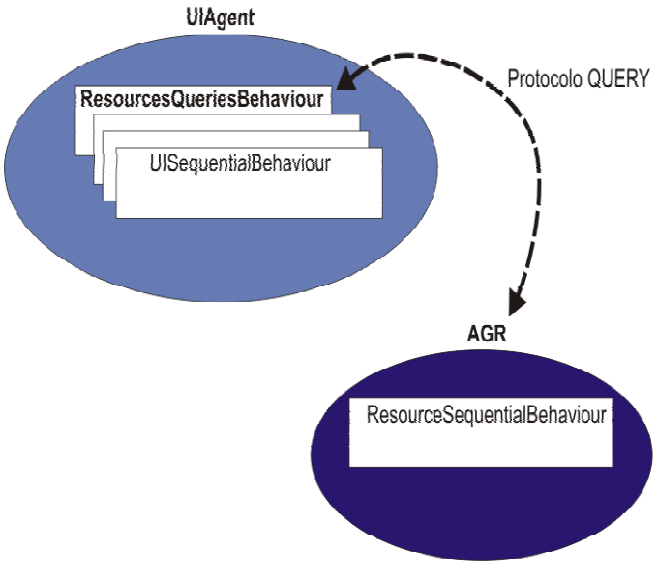


Figura 33. Modelo de comportamiento para preguntas del agente gestor de recursos (AGR)

3.1. Ejemplo de mapeo de un caso de uso

Siguiendo el diseño presentado en el capítulo tres, el siguiente paso es llevar los casos de uso propuestos para cada uno de los agentes, a una implementación haciendo uso de la plataforma escogida. JADE ofrece una serie de funcionalidades para la comunicación entre agentes, que sirven de base para realizar esta implementación.

La comunicación en la plataforma está basada en “intervenciones” entre grupo de agentes, cada una de esta intervenciones es un mensaje que tiene un sobre y un contenido, y que se trasmite por un medio determinado. Cuando se realiza el mapeo, se deben elaborar los mensajes especificados con sus performativas y contenido, y también los mecanismos de control de estos mensajes especificados para cada caso de uso.

La clase que implementa los mensajes se denomina *ACLMessage*. Sus métodos y atributos permiten especificar el agente remitente, los agentes destinatarios, contenido, tipo de lenguaje utilizado, tipo de ontología utilizada y otras características importantes. Esta clase es utilizada por los métodos de envío (*send*) y recepción (*receive*, *blockingReceive*) como argumento, de tal manera que

los agentes puedan acceder a todos los elementos cuando se envía o reciba un mensaje.

En cada comportamiento (*Behaviour*) se puede especificar una estructura para el manejo de mensajes. Para cada agente se especifica un comportamiento principal (*ProcessInfoBehaviour*) que admite una cadena de otros tipos de comportamientos, que pueden reordenarse dinámicamente. El comportamiento puede reiniciar su cadena de eventos en cualquier momento a través del uso del método *reset*.

Para ilustrar la implementación de los casos de uso, se tomará como ejemplo al agente gestor de aplicaciones (AGA, ver figura 34).

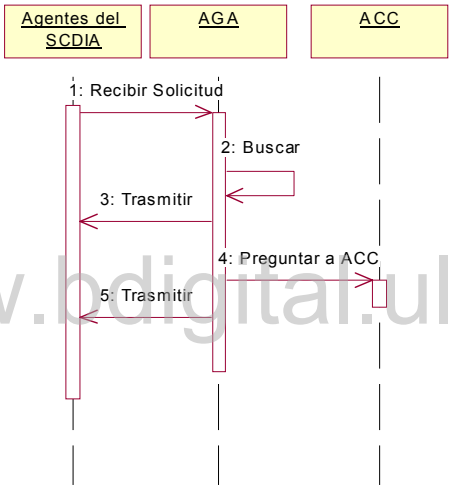


Figura 34. Diagrama de Interacción del AGA

Para este caso, el agente debe poseer un comportamiento que constantemente esté en espera de solicitudes (mensajes), que usen el lenguaje ACL (*Agent Content Language*), y la ontología definida para hablar de recursos (*ResourceOntology*):

```
class AGAGetRequestBehaviour extends SimpleAchieveREResponder
   {

  public AGAGetRequestBehaviour(Agent myAgent){
super(myAgent,MessageTemplate.MatchOntology(ResourceOntology.NAME)))}
```

El comportamiento *AGAGetRequestBehaviour* extiende el comportamiento *SimpleAchieveREResponder*. El agente al cual pertenece el comportamiento es argumento del constructor. En el constructor también se chequea que la ontología del mensaje corresponda con la ontología de recursos.

La clase *SimpleAchieveREResponder* al recibir un mensaje que corresponda con las características señaladas en el constructor (paso 1 del diagrama de interacción) invoca automáticamente el método *prepareResponse* que debe enviar la respuesta al remitente del mensaje iniciador, este proceso corresponde con el paso 3 del diagrama de interacción.

El paso 2 y 4, búsqueda y pregunta al ACC respectivamente, del diagrama de interacción se ejecutan dentro del método *prepareResponse*, este método se invoca al recibir un mensaje con las características especificadas en el constructor del objeto comportamiento. Si el ACC no puede procesar la solicitud envía un mensaje de rechazo al AGA y este lo comunica al agente iniciador:

```
try{
AbsPredicate cs = (AbsPredicate)myAgent.getContentManager().extractAbsContent(msg);
Ontology o = myAgent.getContentManager().lookupOntology(ResourceOntology.NAME);
  if (cs.getTypeName().equals(SLVocabulary.NOT)){
  //… Código que procesa el rechazo de la solicitud
} // Fin del bloque try
                        catch (Codec.CodecException fe) {
                                System.err.println("FIPAException en llenar/extraer Msgcontent:"
+ fe.getMessage());
                        }
                        catch (OntologyException fe) {
                                System.err.println("OntologyException in getRoleName:" +
fe.getMessage());
                        }
```

En el código anterior se verifica que el tipo de ontología es el correcto, se extraen el contenido del mensaje (*extractAbsContent*) y se chequea si corresponde a un rechazo (*NOT*). De ser cierto esto, se procesa este rechazo. También se incluyen el procesamiento de las excepciones posibles: *CodecException* que tiene que ver con los errores producidos por la codificación

del contenido, y *OntologyException* que tiene que ver con el chequeo de la ontología.

3.2. Comunicación entre agentes

La comunicación entre agentes está basada en un sistema de pase de mensajes, que representan actos de habla entre agentes. Cada agente posee una cola privada de mensajes, la cuál puede ser accedida diferentes formas, permitiendo una comunicación síncrona o asíncrona. Los mensajes contienen dos partes, el sobre (*envelope*) y el contenido (*content*). Todos los mensajes deben poseer remitente, destinario, esquema de codificación, protocolo de coordinación, protocolo de transporte y ontología. El contenido puede ser texto plano o estar escrito en lenguaje ACL (Agent Comunication Language).

El lenguaje ACL está disponible para el SGS, admite performativas y uso de ontologías. También se provee un conjunto de clases para implementar protocolos de coordinación.

La capa de transporte, que sirve de base para el sistema de pase de mensajes, puede ser de distinto tipo según la plataforma donde se instale el sistema multiagente. Están disponibles las implementaciones bajo Java RMI, CORBA y el protocolo HTTP. También es permitido agregar otro tipo de implementación para la capa de transporte, solo que ésta debe proveer ciertos servicios para el nivel superior.

Existen varios métodos para la gestión de la computación que implementa la clase *Agent*, lo más importantes se listan a continuación:

send(ACLMessage msg): envía el mensaje *msg* al destinatario especificado en el sobre.

ACLMessage receive(): obtiene el primer mensaje que se encuentra en la cola.

ACLMessage blockingReceive(): obtiene el primer mensaje que se encuentra en la cosa, este método a diferencia del anterior suspende la ejecución hasta que un mensaje esté disponible en la cola. Es bloqueando y permite comunicaciones síncronas.

ACLMessage receive(MessageTemplate pattern, long time): obtiene el primer mensaje de la cola que coincida con el patrón especificado. El segundo parámetro

indica el tiempo máximo que espera el método expresado en milisegundos antes de seguir la ejecución.

La clase que implementa el concepto de mensaje es *ACLMessage*, ésta clase contiene un conjunto de atributos y métodos para la gestión. Con esta clase es posible crear mensajes de respuestas, agregar múltiples destinatarios, llenar los campos del sobre y el contenido, utilizar ontologías y ejecutar acciones de copia y guardado.

La comunicación entre los agentes del SGS está basada en esta implementación. Se debe acoplar el esquema de comportamiento secuencial de cada uno de los agentes con el envío y recepción de mensajes.

Existen comportamientos configurados para esperar y procesar determinados tipo de mensaje, y son utilizados para implementar los diferentes protocolos de coordinación. Por ejemplo el comportamiento *SimpleAchieveREResponder* es utilizado en los agentes de capa de servicios para esperar el mensaje de pregunta del protocolo *FIPA-QUERY*:


```
class HandleResourcesQueriesBehaviour extends SimpleAchieveREResponder
   {

  public HandleResourcesQueriesBehaviour(Agent myAgent){
super(myAgent, MessageTemplate.and(
MessageTemplate.MatchProtocol(FIPANames.InteractionProtocol.FIPA_QUERY),
                        MessageTemplate.MatchOntology(ResourceOntology.NAME)))}


        public ACLMessage prepareResponse(ACLMessage msg){
           ACLMessage reply = msg.createReply();
            …
               send(reply);
```

El código anterior define un comportamiento para el agente gestor de recursos (AGR) basado en la espera de un determinado mensaje, que coincida con el protocolo *FIPA-QUERY* y que utilice la ontología *ResourceOntology*. Cuando se recibe el mensaje con estas características, se ejecuta el método *prepareResponse*, este método crea un mensaje de respuesta, llena su contenido en función de la pregunta que se realice, y lo devuelve al agente o agentes que realizaron la pregunta.

De tal manera, cuando se requiere utilizar los protocolos de coordinación se debe definir comportamientos que implementen la comunicación y los diferentes actos de habla que se especifican para el protocolo.

3.3. Ontologías

Un aspecto importante que se debe considerar cuando se desarrollan SMAs, tiene que ver con la comunicación. La comunicación, como en las sociedades humanas, tiene una gran importancia en las comunidades de agentes. Cuando se produce la comunicación entre seres humanos, todo lo que se habla es una representación de los objetos reales, son relaciones de esos objetos con otros objetos o con los propios seres humanos. Esta representación tiene su equivalente en los SMAs, y se expresa a través de las ontologías. El aspecto ontológico de la comunicación es representado en los SMAs a través de conceptos, acciones, predicados y secuencias, que tienen su implementación en objetos que pueden ser conocidos en las intervenciones de cada agente.

En este sentido, para el SGS se desarrolló una ontología denominada "resource-ontology", que agrupa los elementos ontológicos que tienen que ver con la gestión de recursos, ya sean estos recursos de hardware, de software, de datos o de comunicación. Por ejemplo, se definieron algunos conceptos y predicados que tienen que ver con los recursos:

- *Memory*: memoria del computador (clase base), concepto
  - *PhisicalMemory*: memoria física (RAM) , concepto
  - *SwapFile*: memoria de intercambio, concepto
  - *VirtualMemory*: Memoria virtual, concepto
- *Drive*: unidad de disco, concepto
- *Processor*: procesador del computador, concepto
- *Display*: Pantalla, concepto
- *Resolution*: resolución de la pantalla, concepto
- *Modem*: modem de comunicaciones, concepto
- *Printer*: Impresora , concepto
- *NodeInfo*: información de los recursos de hardware, predicado

Cada uno de estos elementos representa un concepto, predicado, o una secuencia de la ontología. En la implementación, los elementos son objetos "entendibles" por los agentes del SGS. En cada intervención los agentes pueden

hacer uso de estos elementos, que en conjunción con la performativas dan lugar a acciones, informaciones, preguntas y respuestas. La ontología permite representar el conjunto de elementos de habla o vocabulario que tienen significado para los agentes, es decir, que se encuentran vinculados al ambiente.

Para la implementación de la ontología de recursos se hace una extensión de la clase *Ontology* del paquete de clases de JADE (ver figura 35). Los elementos de la ontología (conceptos, predicados, acciones, términos) son clases definidas separadamente que tienen un vínculo con la ontología de recursos a través del uso del método *add* y las clases *Schema*.

Figura 35. Extensión de la clase Ontology

El siguiente código demuestra la forma en que se realiza el vínculo entre los elementos y la ontología de recursos:

```
import jade.content.onto.*;
import jade.content.schema.*;

public class ResourceOntology extends Ontology {

 //NAME
 public static final String NAME = "Resource-Ontology";
 // The singleton instance of this ontology
 private static Ontology theInstance = new ResourceOntology();
 public static Ontology getInstance() {
    return theInstance;
 }
 //VOCABULARY
```

```
 public static final String DRIVE="Drive";
public static final String DRIVE_FILESYSTEM="filesystem";
…
//Constructor
private ResourceOntology(){
     super(NAME, BasicOntology.getInstance());
   try {

    // added Concept(s)
    ConceptSchema driveSchema = new ConceptSchema(DRIVE);
    add(driveSchema, Drive.class);
}
```

En las primeras líneas del código anterior se incluyen los paquetes *onto* y *schema* que son necesario para el manejo de ontologías; seguidamente se extienda la clase *Ontology* y se describe el vocabulario, incluyendo el nombre de la ontología (*Resource-Ontology*). En el constructor de la clase *ResourceOntology* se utiliza el método *add* para vincular los conceptos, términos y demás elementos a la ontología de recursos.

3.4. Acceso a recursos

Para realizar el acceso a recursos como se especifica en el diseño conceptual, se implementó una vinculación con los componentes. JADE permite interacción con JavaBeans, y CORBA; pero no tiene un enlace directo con los componentes COM. Es por esto que se especifica una forma de realizar un proxy COM-Java utilizando JNI (Java Native Interface) para integrarlo a la ontología de recursos especificada para la capa de gestión de servicios.

En primer lugar, es necesario identificar el componente COM que se requiere integrar, este proceso de identificación consiste en conocer la interfaz que exporta el componente, y que debe estar contenida en un archivo de extensión *tlb* (type library binary). Este archivo debe estar presente en el directorio donde se encuentran los archivos de sistema (%WINDIR%/*System32*) o en el directorio de instalación del componente. Si se utiliza un compilador C++ la librería se enlaza mediante la siguiente directiva:

```
#import      "c:\Windows\System32\COMTest.tlb"      raw_interfaces_only
named_guids
```

Esta directiva le indica al compilador la ruta y nombre del archivo de interfaces para el componente COM que se requiere utilizar. Generalmente el desarrollador del componente incluye un archivo de ayuda o manual del programador, que describe las funcionalidades del componente.

Luego es necesario construir un *proxy* de enlace dinámico (biblioteca DLL) que realice el vínculo entre el componente COM y el SGS.

Para implementar este *proxy* se puede utilizar el compilador Visual C++[20]. El JDK provee los archivos necesarios que contienen los tipos de datos y demás recursos para construir la biblioteca. El código siguiente ilustra como definir un método en la biblioteca:

```
JNIEXPORT void JNICALL Java_JNISpace_setCOMSize(JNIEnv *env, jobject canvas, jint
width, jint height)
{
        if (m_spMyCOMObject != NULL)
        {
                m_spMyCOMObject->put_Width((long)width);
                m_spMyCOMObject->put_Height((long)height);
        }
}
```


La palabra *JNIEXPORT* define que es un método para ser accedido desde Java, *JNICALL* define la forma en que serán leídos los parámetros de tal manera que coincida con el formato JNI; seguidamente se declara el nombre del método (setCOMSize), el prefijo del método indica que pertenece a la clase JNISpace. Los parámetros corresponden con el entorno en Java, la interfaz canvas donde será mostrado el objeto COM, y los parámetros del método respectivamente.

El método que se muestra es un envoltorio para un objeto COM con interfaz gráfica. El objeto COM se denomina *m_spMyCOMObject*. Dentro del envoltorio es posible acceder a cualquiera de sus métodos o propiedades.

Para ser accedido este método desde Java solo es necesario cargar la biblioteca de enlace dinámico y declararlo como método nativo:

```
class JNISpace extends Canvas {
static {
                System.loadLibrary("JNISpace");
        }
...
public native void setCOMSize(int width, int height);
...
```

Luego es posible invocar el método nativo desde Java. El proceso de "envolver" los objetos COM puede ser automatizado a través de una utilidad que genere la biblioteca de enlace dinámico (proxy) y el código en Java.

Los componentes son vinculados con los conceptos de la ontología interfaz utilizando reflexión. La reflexión permite inspeccionar los atributos, métodos y parámetros de las clases descritas en Java, por lo tanto, es posible utilizar esta característica para examinar los dos objetos y hacer un emparejamiento entre ellos.

```
matchComponent(pred, component);
getContentManager().fillContent(reply, pred);
```

El método *matchComponent* vincula las propiedades de los componentes con los atributos de los elementos de la ontología, para realizar esto se inspecciona las propiedades del componente (*component*) y se busca sus correspondientes atributos en el objeto de la ontología (*pred*), también es posible realizar el proceso inverso utilizando este método estático.

El código anterior ilustra el uso de *matchComponent*, la primera línea llama al método estático de enlace entre el elemento de la ontología y el componente u objeto. La segunda línea transforma el elemento de la ontología y lo escribe en el contenido del mensaje *reply*, que luego será enviado utilizando el método *send*.

Para integrar otro tipo de componente al SGS se realiza el mismo procedimiento descrito anteriormente. En el caso de los componentes JavaBean, no es necesario armar el proxy con la aplicación nativa, debido a que estos componentes están escritos en Java, solo es necesario disponer del J2EE (Java 2 Enterprise Edition).

CAPITULO V: Ejemplo del sistema de gestión de servicios

Para mostrar las características del sistema de gestión de servicios se ejemplifica una aplicación de optimización, que generalmente está incluida dentro del rango de aplicaciones de una plataforma de automatización.

1. Descripción del problema

El entrenamiento de redes neuronales [4,11,12] con la finalidad de identificar o reconocer patrones que permitan predecir valores, en el ámbito computacional, se transforma en una aplicación de software que requiere un uso importante de recursos, así como también un tiempo considerable del usuario o "entrenador". En algunos casos es inmanejable la ejecución de esta aplicación, por ejemplo, cuando existe una cantidad considerable de datos a procesar. Una red neuronal es un modelo matemático basado en las redes neuronales biológicas, que tienen un proceso de "aprendizaje" que consiste en el ajuste de valores denominados pesos a través de la aplicación de determinado algoritmo, entre los cuáles se puede nombrar uno de los más importantes como es el de retropropagación.

En el ambiente industrial, en ocasiones es necesario identificar procesos físicos, que se realizan a diario en alguna de las plantas del complejo. Debido a que es difícil elaborar un modelo matemático exacto o aproximado siguiendo las leyes físicas del proceso, generalmente se plantea la construcción de un modelo basado en redes neuronales que utilice datos históricos del proceso estudiado para usarlo en el entrenamiento de la red neuronal. Se tomó como ejemplo un reactor biológico: se tiene un conjunto de datos históricos de entrada y salida. Dado que uno de los objetivos del SGS es administrar eficientemente los recursos y aplicaciones de una plataforma de automatización, se toma ésta aplicación y se implementa utilizando el SGS.

2. Descripción del ejemplo

La aplicación se corre sobre el sistema de gestión, consiste en el procesamiento de datos numéricos, discretos y/o continuos que pueden ser recolectados desde un sistema de histórico de datos y/o un sistema SCADA (Supervisory Control And Data Adquisiction: Sistema de adquisición de datos y control supervisorio) de manera que permitan elaborar un modelo de matemático de identificación para un proceso determinado. Esta tarea generalmente consume una importante cuota de recursos, en el ejemplo se realiza de forma distribuida, se gestionan los recursos y aplicaciones basado en el modelo de inteligencia de los agentes gestores de recursos y aplicaciones. En otras palabras, el sistema utiliza los diferentes recursos y aplicaciones disponibles para llegar a la meta propuesta. Para el ejemplo, el usuario dispone de una interfaz gráfica para realizar la introducción de datos, aunque los datos podrían eventualmente ser obtenidos desde el sistema histórico o SCADA.

La aplicación puede utilizar diferentes esquemas para hallar el conjunto de reglas y patrones que satisfagan las condiciones del problema. Entre estos esquemas están las redes neuronales.

Para el ejemplo tratado se escogió una red neuronal de retropropagación. La interfaz de usuario de está aplicación es un agente, por lo tanto posee todas las cualidades de los agentes del SGS. Al usuario se le muestra un cuadro de diálogo para que ingrese la ruta del archivo que contiene la configuración de la red neuronal y el archivos de datos (ver figura 36),

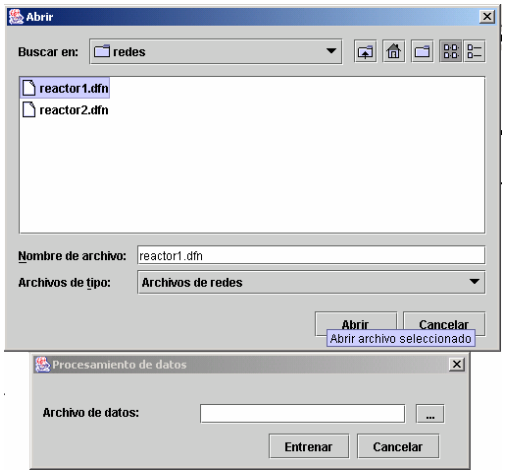


Figura 36. Cuadro de diálogo para entrenamiento de la red neuronal

Los agentes que pertenezcan al sistema, pueden poseer interfaces gráficas, las cuáles permiten interacción con los usuarios, o también llamados agentes humanos. La interfaz permite introducir los datos necesarios para realizar el entrenamiento de la red neuronal y recibir el modelo de predicción del problema tratado.

Los agentes que tienen interfaz gráfica deben ser extensión de la clase *GuiAgent*. Esta clase contiene dos métodos para controlar los eventos generados desde la interfaz de usuario: *OnGuiEvent* y *PostGuiEvent*(ver figura 37). Los dos métodos administran la cola de eventos, pero la diferencia es que el primero es abstracto (debe ser implementado en la clase extensión) y el segundo no, por lo cuál, tiene su implementación en la clase base y puede ser invocado de la clase extensión para enviar un evento especifico. La interfaz debe tener una estructura basada en el paquete *awt* de la biblioteca de clases de Java.

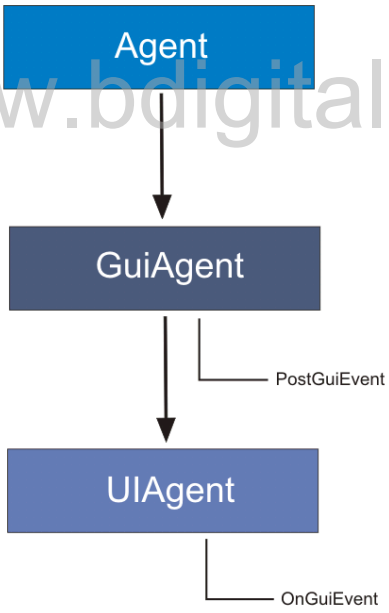


Figura 37. Jerarquía de clases para la Interfaz gráfica

A través de la interfaz se escoge el archivo que contiene la configuración de la red con extensión de “dfn”, y automáticamente también se cargan los datos, que deben estar almacenados en un archivo con el mismo nombre del archivo de

configuración, pero con extensión “dat”. En el archivo de configuración se especifican los nombres y tipos de variables de entrada y salida, además de los parámetros de entrenamiento, y en el archivo de datos se especifican en filas los patrones para entrenar la red:

```
// proceso.dfn
continuous entrada1u
continuous saliday
coef  0.1
momen 0.7

// proceso.dat    -------> Data
1.2 3
2.1 5
6.7 6
```

En el archivo de configuración de la red también se escriben los parámetros del entrenamiento, tales como el coeficiente de aprendizaje (*coef*) y el momento (*momen*), realizando las validaciones necesarias. El coeficiente de aprendizaje es un valor numérico que ajusta el grado en que los pesos son modificados, por lo tanto, se dice que en este mismo grado la red aprende, por su parte, el parámetro “momento” permite a la red hallar en teoría, mejores errores.

3. Funcionamiento del SGS

Al presionar el botón “Aceptar” se inicia el proceso de gestión distribuida para el procesamiento de la información recolectada. En este proceso participan los agentes AGR, AAA; AGA y los agentes, recursos y aplicaciones que sean necesarios y estén disponibles para lograr los resultados esperados (Figura 38).

Figura 38. Sistema multiagente para entrenamiento de redes neuronales

Para la comunicación entre agentes se hace uso de la ontología de recursos (*Resource-Ontology*) y de la ontología FIPA para movilidad (*FIPA-Mobility-Ontology*), que incluye los conceptos, predicados y acciones para llevar a cabo cada tarea.

La aplicación se inicia realizando un proceso de negociación entre agentes. Este proceso tiene como finalidad seleccionar cuáles nodos de la red reúnen las condiciones para ejecutar las tareas de procesamiento, es decir, que nodos tienen los suficientes recursos de memoria principal y secundaria, procesadores, recursos gráficos, y conexión de la red, para ejecutar la aplicación asignada. En este proceso participa el agente gestor de recursos (AGR), el agente administrador de agentes (AAA) y los agentes que se encuentran dispersos en los otros nodos de la red.

En primer lugar, el AGR solicita al AAA una lista de agentes que contenga un agente por nodo, de tal forma de poder realizar la consulta a estos agentes sobre la información de los recursos del nodo donde se encuentran operando. Como todos los agentes contienen el tipo de comportamiento *ResourcesQueriesBehaviour* pueden responder al AGR sobre los recursos locales, de tal manera que el AGR reúna información los recursos de cada nodo del sistema. El AGR hace uso del componente inteligente para evaluar si un nodo es apto para procesar cierta carga de trabajo, y de esta manera dar respuesta a la petición hecha. Es entonces cuando se inicia un proceso de coordinación basado en un protocolo *QUERY*, que hace una evaluación de recursos de cada nodo y realiza un contrato con los agentes que se encuentran en los nodos que cumplen con las condiciones necesarias para realizar la tarea de procesamiento. El AGR inicia una intervención preguntando sobre la información del nodo. En esta intervención se envía un predicado *NodeInfo* descrito en la ontología de recursos.

Cuando este mensaje llega al nodo destino se hace un proceso de emparejamiento con un componente nativo perteneciente a la plataforma, a través del uso del método estático *matchComponent* (ver figura 39), y se envía el mensaje de respuesta informando sobre la característica de cada nodo, luego el AGR decide los nodos que realizarán el procesamiento.

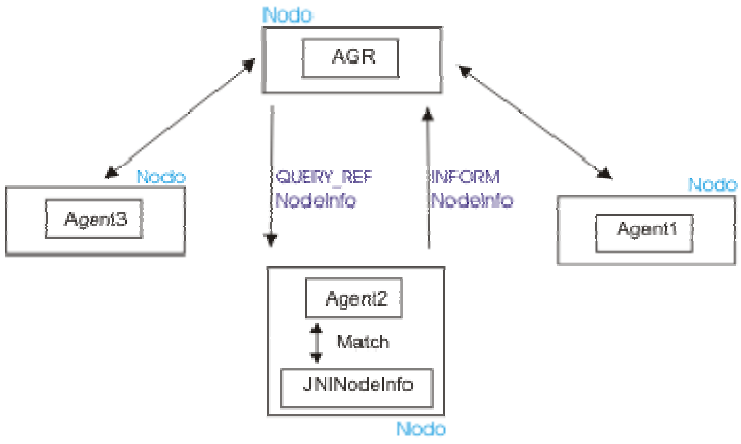


Figura 39. Coordinación del AGR.

Por su parte, el Agente Gestor de Aplicaciones (AGA) realiza el trabajo de elección del conjunto de agentes especializados que pueden procesar los datos y devolver los resultados esperados. Como se explicó anteriormente, uno de los esquemas más utilizados para este tipo de procesamiento son las redes neuronales, pero existen otros esquemas tales como los árboles de decisión y métodos de aproximación que también pueden ser aplicados. Además de poder aplicar diferentes tipos de procesamiento, también es posible utilizar diferentes tipos de implementación, para el ejemplo, se utilizan dos tipos de implementaciones para redes neuronales: una desarrollada en Java y otra basada en un componente COM que se comunica con MATLAB® (una aplicación comercial que provee herramientas para el cálculo matemático)[6]. Cuando se consulta las aplicaciones disponibles el AGA sugiere estos dos agentes que están vinculados a dos aplicaciones: NNAgent y NNMatlabAgent (ver figura 40).

[6] Para mayores detalles de esta aplicación puede consultar la dirección web: http://www.mathworks.com

Figura 40. Agente Gestor de aplicaciones (AGA)

El agente NNAgent puede ejecutarse en los nodos seleccionados por el AGR, en cambio el agente MatlabAgent debe ejecutar el procesamiento en el nodo donde se encuentra instalada el componente COM.

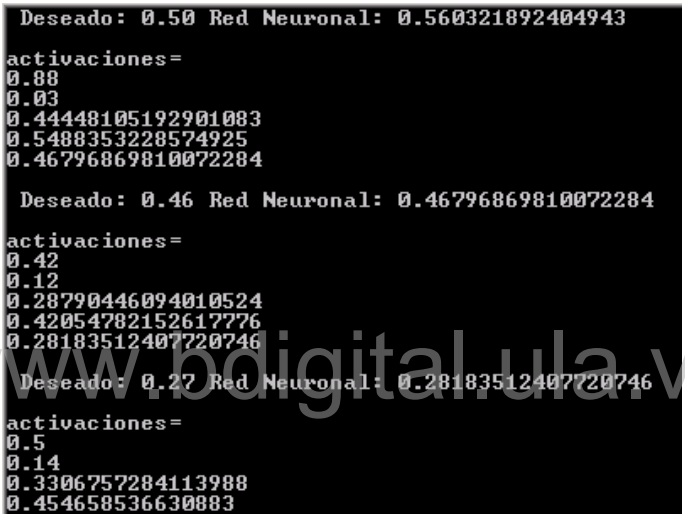


Figura 41. Resultados del entrenamiento con la red neuronal

Luego del procesamiento los resultados son devueltos al agente interfaz (UIAgent) y finalmente mostrados al usuario (ver figura 41).

Capítulo VI. Conclusiones

La programación multiagente brinda una nueva posibilidad para el desarrollo de sistemas para computadoras. El concepto de *agente* ofrece un marco de trabajo que permite utilizar características que han estado presentes desde hace años en el campo de la computación, pero que por su complejidad ha resultado difícil de integrar en una única entidad. De estas características atribuidas a los agentes, las principales son la autonomía, la movilidad, la inteligencia y la comunicación.

En este trabajo se desarrolló una capa de software para dar apoyo a sistemas multiagentes, que brinda un conjunto de servicios básicos y extensibles para la implementación de sistemas multiagentes en plataformas de automatización.

En el proceso de desarrollo, se utilizó la metodología MAS-CommonKads extendida, que facilitó el proceso de descripción de las actividades y tareas de los agentes, así como también permitió la descripción de la arquitectura del sistema.

Para la implementación, se extendió la plataforma JADE que cuenta con un conjunto de clases escritas en el lenguaje JAVA para el desarrollo de sistemas multiagentes. Se implementaron los agentes del Sistema de Gestión de Servicios: Agente *Administrador de Agentes*, Agente *Gestor de Recursos* y Agente *Gestor de Aplicaciones*. El tipo de agente *Base de Datos* y el tipo de agente de *Control de Comunicación* fueron implementados parcialmente, ya que resta el desarrollo de ejemplos con plataformas reales tales como sistemas de información en tiempo real y otros sistemas multiagentes respectivamente.

Se desarrollo un esquema para integrar el sistema multiagente con los recursos y aplicaciones de la plataforma computacional. Se propuso un tipo de integración que utiliza conceptos tales como componentes y reflexión, que permite integrar elementos ontológicos con componentes reales dentro la plataforma, posibilitando así, la inclusión rápida y uniforme de los recursos y aplicaciones disponibles.

EL SGS fue probado con un ejemplo de procesamiento computacional distribuido, donde se muestra el uso de características tales como autonomía, comunicación, inteligencia y movilidad de los agentes. Estas características permiten elaborar un modelo de aplicación que utilice los recursos y aplicaciones disponibles dinámicamente de forma, que se utilicen eficientemente, y que conlleve al logro de las metas asignadas.

El SGS fue desarrollado en función de servir de apoyo a plataformas de automatización, y concretamente al SCDIA, donde existe una heterogeneidad en las aplicaciones y recursos disponibles, pero puede también ser utilizado en plataformas computacionales de distinta naturaleza, donde sea necesario la gestión inteligente y dinámica de los recursos y aplicaciones.

Dado que sólo se provee un conjunto de clases para extender y agregar tipos de agentes, y de herramientas visuales para la administración de agentes, se recomienda el desarrollo de una aplicación con interfaz gráfica que facilite la construcción de tipos de agentes. Este proceso actualmente solo puede llevarse a cabo a través de la escritura en un archivo de texto de la clase correspondiente, que después de compilado sea integrado al sistema.

BIBLIOGRAFÍA

[1] Aguilar, Jose, Cerrada, Mariela et al. **Aplicación de Sistemas Multiagentes a Problemas del Mundo Real**. XXVII Conferencia Latinoamericana de Informática (CLEI). 2001.

[2] Aguilar, Jose, Cerrada, Mariela, et al. **Applications of the Agents Referente Model for Intelligent Distributed Systems**. in: Advances in Systems Science: Measurement, Circuits and Control. Edited by: N.E. Mastorakis and L.A. Pecorelli. 2001.

[3] Bellifemine, F,, A. Poggi & G. Rimassi, **JADE: A FIPA-Compliant agent framework**, Proc.Practical Applications of Intelligent Agents and Multi-Agents, Abril 1999. (*Ver http://sharon.cselt.it/projects/jade)*

[4] Bigus Joseph & Jennifer Bigus. **Constructing Intelligent Agents using Java**. Wiley Computer Publishing. 2001.

[5] **FIPA Agent Management Specification**. The Foundation for Intelligent Phisical Agent. California. USA. 2002.

[6] **FIPA QUERY Interaction Protocol Specification**. The Foundation for Intelligent Phisical Agent. California. USA. 2002.

[7] **FIPA CONTRACT-NET Interaction Protocol Specification**. The Foundation for Intelligent Phisical Agent. California. USA. 2002.

[8] **FIPA ACL Message Structure Specification**. The Foundation for Intelligent Phisical Agent. California. USA. 2002.

[9] **FIPA Agent Message Transport Service Specification**. The Foundation for Intelligent Phisical Agent. California. USA. 2002.

[10] Griss, Martín, Fonseca Steven, Cowan Dick y Kessler Robert. **SmartAgent: Extending the JADE Agent Behaviour Model**. AOSE Workshop. Orlando Florida, Julio, 2002.

[11] Hagan M, Demut H y Beale M. **Neural Network Design**. Publishing Company. 1996.

[12] Haykin, S. **Neural Networks: a comprehensive foundation**. IEEE Computer Society Press. 1994.

[13] Iglesias, Carlos**. Definición de una metodología para el desarrollo de sistemas multiagente**. Universidad Politécnica de Madrid. 1998.

[14] Moussalli, Gloria. **Modelo de Referencia para el desarrollo de Sistemas de Control Distribuidos Inteligentes basados en Agentes**. ULA. 2000.

[16] Nelson, Jeff. **Programming Mobile Objects with Java**. WILEY. 1999.

[17] Nillsson, Nils. **Inteligencia Artificial, Una nueva síntesis**. Mc Graw Hill. 1998. España.

[18] Rivas, Francklin, Aguilar, José, et al. **Tercer Informe Técnico Agenda Petróleo**. Universidad de los Andes. 2002.

[19] Rusell Stuart y Norvig Peter. **Inteligencia Artificial. Un enfoque moderno**. Prentice Hall. 1995. México.

[20] Spell, Brett. **Professional Java Programming**. Wrox Press. 2001.

[21] Tanenbaum, Andrew. **Sistemas Operativos Distribuidos**. Prentince Hall. 1ra Edición. 1995. México.

[22] Templeman´s Julian. Beginning MFC COM Programming. Wrox Press. 1997.

[23] Weiss, Gerhard. **Multiagent Systems**. The MIT Press. 1999.

APENDICE A: Manual de usuario JSCDIA

1. Instalación

Para instalar el sistema de gestión de servicios JSCDIA (SCDIA bajo JADE) descomprima sobre el directorio de su selección el archivo jscdia.zip o jscdia.tar.gz que se encuentra en el disco de instalación, por ejemplo:

```
home/cemisid>tar –xzf jscdia.tar.gz
```

Cambia el directorio “jade/jscdia”:

```
home/cemisid>cd jade/jscdia
```

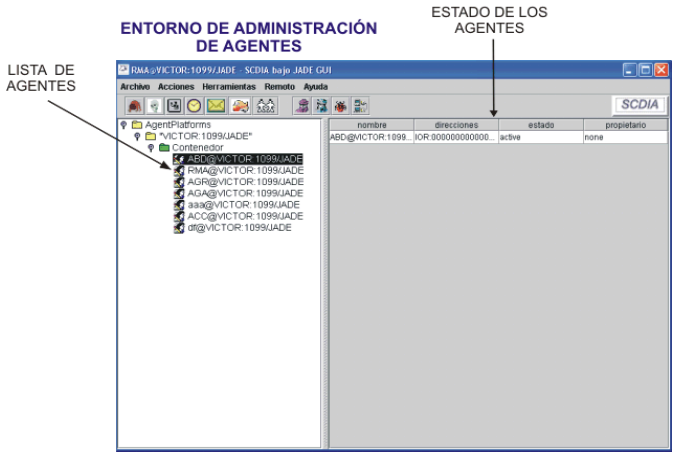
Desde este directorio puede ejecutar las aplicaciones del JSCDIA para cada nodo instalado.

2. Administración del SGS

Con el SGS se provee una aplicación gráfica para la administración del sistema multiagente que puede ser iniciada desde la línea de comando tecleando:

```
home/cemisid>jscdia -gui
```

La opción “-gui” indica que se levante la aplicación gráfica. Debe aparecer la interfaz de la aplicación (ver figura 1).

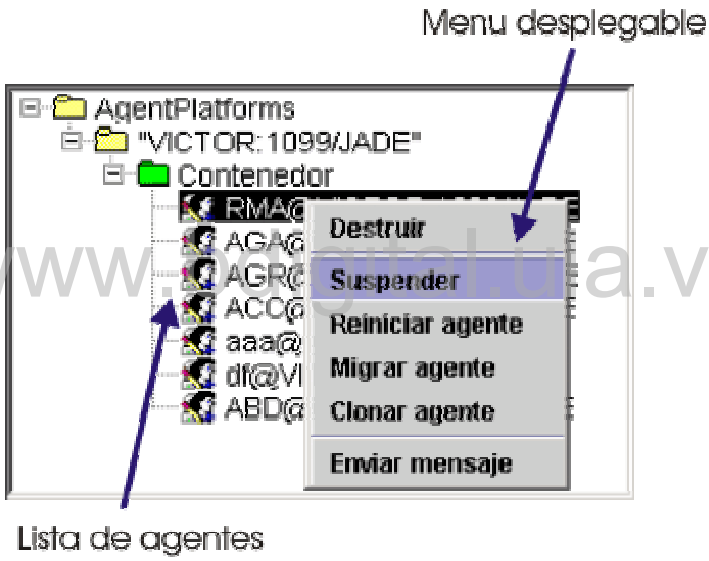


Apéndice A. Figura 1. Aplicación para la administración del SGS

En el lado izquierdo de la pantalla aparecen la lista de agentes que forma parte del SGS: Agente Administrador de Agentes (AAA), Agente Gestor de Recursos (AGR), Agente Gestor de Aplicaciones (AGA), Agente de Base de Datos (ABD) y Agente de Control de Comunicación (ACC). Se indica que pertenecen al nodo de la red identificado como *VICTOR*, que las comunicaciones se realizan a través del puerto 1099 utilizando el protocolo TCP/IP y que se está utilizando la plataforma "JADE".

En lado derecho de la pantalla aparecen los atributos del agente seleccionado: nombre completo, estado, propietario, y dirección de la red.

En lado superior se encuentra el menú y la barra de herramientas.



Apéndice A. Figura 2. Menú de comandos sobre el elemento agente

Seleccionando un elemento del árbol del lado derecho de la pantalla y pulsando el botón derecho aparece un menú (ver figura 2) que permite realizar un conjunto de funciones sobre él.

Los agentes pertenecen a un contenedor y pueden moverse entre contenedores que pueden estar localizados en el mismo computador o en otros nodos de la red.

Es posible iniciar aplicaciones de JADE desde el menú de usuario, estas aplicaciones se muestran en el menú de Herramientas (ver figura 3).

Apéndice A. Figura 3. Aplicaciones de JADE.

Las aplicaciones disponibles son:

Sniffer: permite visualizar gráficamente las intervenciones y pase de mensajes entre agentes.

DummyAgent: inicia un agente de ejemplo que no tiene ninguna funcionalidad

DFGUI: muestra las propiedades del directory facilitator (df). El df un agente que registra los servicios y propiedades de los agentes y ofrece servicios de búsqueda y filtros sobre esta información.

Instropector Agent: esta aplicación permite inspeccionar el estado y los comportamientos de los agentes, así como también los mensajes que procesa.

3. Inicializar un contenedor remoto

La plataforma permite inicializar remota o localmente un contenedor y bajo este contenedor activar un conjunto de agentes. Si existe un contenedor local se puede iniciar otro contenedor remoto que dependa del AAA local haciendo uso de la siguiente sintaxis:

```
home/cemisid>jscdia –host <nombredelhost> -container <listadeagentes>
```

El parámetro *nombredelhost* es el nombre del nodo de la red al que se quiere conectar y el parámetro *listadeagentes* es el conjunto de agentes con su nombre y tipo (nombre:tipo) que se desea inicializar, por ejemplo:

```
home/cemisid>jscdia –host VICTOR -container uiagent:UIAgent
```

La línea de comando anterior inicializa un nuevo contenedor el nodo de red denominado *VICTOR*, que contiene un agente llamado *uiagent* del tipo UIAgent.

4. Creando nuevos tipos agentes

Para crear un nuevo tipo agente y integrarlo al SGS, se debe desarrollar la clase que extienda las propiedades de la clase *Agent* y que utilice la ontología de recursos, de movilidad y de gestión, con la finalidad de comunicarse con los agentes del SGS. Estas clases deben estar en el *classpath* del entorno Java.

Es posible utilizar el método *matchComponent* en el código de los agentes. Para esto es necesario construir los envoltorios JNI y Java en el caso de la interfaz COM, para el caso de los JavaBeans o clases propias de Java no es necesario ejecutar este paso.

APENDICE B: Descripción de JADE (Java Agent Development Framework)

1. Descripción

JADE es un framework que permite desarrollar sistemas multiagentes que cumplan con las especificaciones FIPA (Foundation for Intelligent Phisical Agent). Es un software que está totalmente implementado en el lenguaje Java. Contiene un conjunto de clases que facilita el proceso de desarrollo y depuración de sistemas multiagentes. La plataforma puede ser distribuida en varias máquinas (que necesariamente no ejecutan el mismo sistema operativo) y la configuración y administración puede realizarse remotamente. El sistema requiere Java Run Time versión 1.2.

La arquitectura para la comunicación ofrece mensajería eficiente y flexible. JADE crea y maneja colas de mensajes ACL, colas privadas para cada agente a las cuáles pueden acceder usando distintos procedimientos: por tiempo, en estado bloqueado, basado en un patrón de mensaje, etc.

JADE está siendo utilizado por un grupo de compañías y grupos académicos que pertenecen a la FIPA, y otros que no son miembros, tales como BT, CNET, Universidad de Helsinsky, INRIA, y otros, y recientemente está disponible bajo licencia de código libre (LGPL).

2. Clases Importantes

2.1. jade.core.Agent:

La clase *Agent* es la clase común para definir agentes de software. Provee los métodos para desempeñar las tareas básicas, tales como:

- Pase de mensajes utilizando objetos *ACLMessage*, con despliegue unicast y multicast, admitiendo patrones de mensajes.
- Ciclo de vida del agente soportado, incluyendo inicialización, suspensión, y destrucción del agente.
- Planeación (Scheduling) y ejecución de actividades concurrentes.
- Interacción simplificada con sistemas multiagentes FIPA.

Constructor por defecto:

Agent()

Algunos métodos:

AddBehaviour(Behaviour b): añade un comportamiento a el agente.

RemoveBehaviour(Behaviour b):

blockingReceive(): extra un *ACLMessage* desde la cola de mensajes, el método es bloqueante, es decir, espera hasta que exista un mensaje en la cola,

receive(); extrae un ACLMessage desde la cola de cola.

send(ACLMessage msg): envía un mensaje a otro(s) agentes.

setup(): en este método tipo *protected* se debe escribir el código de inicialización del agente.

2.2. jade.lang.acl.ACLMessage:

Esta clase implementa el Mensaje ACL que cumple la especificación FIPA 2000 “FIPA ACL Message Structure Specification” . Todos los valores pueden ser colocados utilizando los métodos “set” y leídos utilizando los método “get”.

EL par de métodos *setContentObject* y *getContentObject* permiten enviar objetos de forma serializada sobre el contenido del mensaje.

Constructores:

ACLMessage()

ACLMessage(int perf)

Algunos métodos importantes:

addReceiver(AID a) : añade un dirección para el receptor

setSender(AID s): coloca la dirección del remitente del mensaje

createReply(): crea un mensaje de respuesta del mensaje.

setPerformative(int perf): coloca la performativa del mensaje.

setContent(String c): coloca el contenido del mensaje.

2.3. jade.core.AID:

Esta clase representa el identificador del agente.

Constructores:

AID()

AID(java.lang.String name, boolean isGUID)

Algunos métodos:

setName(String n) : coloca el nombre del agente

getName(): obtiene el nombre del método

2.4. jade.core.behaviours.Behaviour

Es una clase abstracta que es base para los comportamientos. Extienda esta clase directamente si necesita escribir un determinado comportamiento, con necesidades especiales de sincronización.

Contructores:

Behaviour()

Behaviour(Agent a)

Algunos métodos:

action(): corre el comportamiento.

block(): bloquea el comportamiento

done(): chequea si el comportamiento ha finalizado

restart(): reinicia un comportamiento bloqueado()

setAgent(Agent a): asocia el comportamiento con el agente del parámetro.

ÍNDICE

Tabla de Figuras