ADMINISTRACIÓN DE SERVIDORES WEB MEDIANTE AGENTES INTELIGENTES

KENNETH RENE SÁNCHEZ SÁNCHEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
BUCARAMANGA
2006

ADMINISTRACIÓN DE SERVIDORES WEB MEDIANTE AGENTES INTELIGENTES

KENNETH RENE SÁNCHEZ SÁNCHEZ

Proyecto de grado presentado como requisito parcial para optar al título de Especialista en Telecomunicaciones

Tutor
CLARA INÉS PEÑA DE CARRILLO
Profesional División de Servicios de Información

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
BUCARAMANGA
2006

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

TITULO: ADMINISTRACIÓN DE SERVIDORES WEB MEDIANTE AGENTES INTELIGENTES[*].

AUTORES: KENNETH RENÉ SÁNCHEZ SÁNCHEZ[**]

PALABRAS CLAVE: Agentes Inteligentes, Servidores Web, Ontologías.

DESCRIPCION:
La presente monografía tiene como propósito estudiar los Agentes Inteligentes y los servidores Web.

Los agentes inteligentes son sistemas con capacidad para tomar decisiones similares a las tomadas por los seres humanos. Es por ello que hoy en día cobra vital importancia conocer su estructura, funcionamiento, clasificación y características, permitiendo de esta manera crear software de última tecnología con aplicativos en cualquier área de trabajo.

Las ontologías también juegan un papel muy importante ya que constituyen la descripción formal de los conceptos que se encuentran en un área específica de interés. Los Agentes Inteligentes se comunican a través de ontologías.

Al igual que los Agentes Inteligentes, los servidores Web cobran vital importancia actualmente debido a la proliferación del uso de Internet. Por tal razón es indispensable entender completamente su funcionamiento, para que de esta forma los sistemas multiagente puedan tomar las decisiones correctas en el momento apropiado.

Basándose en las características y tipos de agentes, se diseña un sistema multiagente para restablecer el servicio del servidor Web Tomcat de la Universidad Industrial de Santander cuando éste se encuentra caído. Para ello se utiliza JADE como plataforma

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Especialización en Telecomunicaciones. Tutor: Dra. Clara Inés Peña de Carrillo

de desarrollo y se crea una ontología propia de comunicación entre agentes. Todo esto se lleva a cabo mediante la codificación hecha en JAVA de los agentes y ontologías diseñados.

SUMMARY

TITLE: ADMINISTRATION OF WEB SERVERS BY MEANS OF INTELLIGENT AGENTS[*].

AUTHORS: KENNETH RENÉ SÁNCHEZ SÁNCHEZ**

KEY WORDS: Intelligent Agents, Web servers, Ontologies.

DESCRIPTION:

The present monograph has the purpose estudy the Intelligent Agents and the Web Servers.

The Intelligent Agents are systems with capacity to make decisions similar to the seizures by the human beings. It is for that reason that nowadays receives vital importance of knowing its structure, operation, classification and characteristics, allowing this way to create software of last technology with applications in any area of work.

The ontologies also play a very important role since they constitute the formal description of the concepts that are in a specific area of interest. The Intelligent Agents communicate through ontologies.

Like the Intelligent Agents, the Web servers at the moment receive vital importance due to the proliferation of the use of Internet. For such reason he is indispensable to understand his operation completely, so that from this form the systems multiagent can make the correct decisions at the appropriate time.

---

[*] Graduation project

[**] Faculty of physics and Mechanical Engineering, Specialization in Telecommunications. Clara Inés Peña de Carrillo

Being based on the characteristics and types of agents, a system is designed multiagent to restore the service of the Web server Tomcat of the Industrial University of Santander when this one is fallen. For it JADE is used as development platform and is created an own ontology of communication between agents. All this is carried out by means of the codification done in JAVA of the agents and designed ontologies.

INTRODUCCIÓN

Desde que el hombre ha existido en la tierra se ha visto obligado a incrementar su conocimiento para poder dominarla y sacar provecho de todo lo que ésta ofrece. Y en su afán de sobrevivir y obtener resultados cada vez más rápidos y eficientes ha desarrollado técnicas que siglos atrás nunca hubiera imaginado: la invención de la rueda, la imprenta, viajes a la luna, los computadores, el Internet, entre otros.

Estos avances han evolucionado de la mano de la ciencia y actualmente acompañada también por la tecnología. Y es precisamente ésta última la que sitúa a los Ingenieros de Sistemas en un punto muy alto de esta evolución, constituyéndolos como herramienta fundamental de desarrollo de nuevas tecnologías, entre las que se destaca la referida en esta monografía: los Agentes Inteligentes.

Al hablar de Agentes Inteligentes involucramos un sinnúmero de personajes que han estudiado a fondo esta tecnología y que sin lugar a duda han dejado huella en cientos de mentes brillantes que se esfuerzan por mejorar los procesos existentes utilizando a los Agentes Inteligentes para tal fin.

En nuestro entorno de trabajo no se puede desconocer la existencia de esta tecnología, y es por esto que esta investigación pretende ser la semilla de un árbol tecnológico guiado por la luz de los Agentes. Por esta razón se ha enfocado hacia uno de los servicios más importantes que posee la Universidad Industrial de Santander - UIS: el Internet.

Actualmente la mayoría de los usuarios de la UIS utiliza el sitio Web institucional, y es por esta razón que es de vital importancia mantenerlo siempre disponible. En este punto entran a jugar un rol importante los Agentes Inteligentes, ya que mediante la presente monografía se pretende desarrollar un sistema multiagente, guiado por un conjunto de reglas, capaz de mantener activo el sistema y fundamentado en la teoría de Servidores Web y Agentes. Para lograr esto, el documento se ha estructurado de

forma gradual, permitiendo al lector adentrarse poco a poco en la implementación y puesta en marcha del agente.

1. PLANTEAMIENTO DEL PROBLEMA

Debido al auge de servicios ofrecidos en Internet se ha generado un crecimiento enorme de usuarios, lo cual conlleva a la optimización de los procesos encargados de mantener una correcta y rápida comunicación y evitar la pérdida de información, el mal funcionamiento y la indisponibilidad de los sistemas.

Muchas son las causas de estos problemas, como el mal uso de la memoria, su configuración, la concurrencia, entre otros y cuya consecuencia es la pérdida funcional del sistema y su posterior caída[1].

Es por ello que surge la necesidad de minimizar el tiempo de indisponibilidad de los servidores WEB, y es aquí donde se ve la utilidad de un agente inteligente semi-autónomo[2] capaz de detectar este problema y tomar la decisión de ejecutar nuevamente el proceso garantizando la puesta en marcha del sistema cada vez que éste lo crea necesario.

Lo que busca esta investigación es estudiar los conceptos fundamentales de los agentes inteligentes, su alcance, diseño e implementación enfocados hacia la administración de servidores WEB, para demostrar que mediante su uso se puede mantener disponible la mayoría del tiempo los sistemas de información implantados en Internet.

[1] Se denomina caída del servidor cuando el proceso encargado de su funcionamiento finaliza por errores generados en su ejecución.

[2] Agente inteligente: Un agente autónomo es un sistema anidado y parte integrante de un ambiente (environment) que detecta o percibe (percepts) datos ambientales, momento a momento, y actúa sobre él con la intención de usar (actions) esos datos para su propia tarea (task) o *agenda*, afectando así lo que va a detectar en el futuro, sin intervención de terceras partes (basado en Franklin y Greasser, 1996).

2. JUSTIFICACIÓN

Para encontrar la utilidad de esta investigación es importante conocer el entorno social y tecnológico del problema.

Para ello es importante resaltar los servicios ofrecidos en la página WEB de la Universidad Industrial de Santander: consulta de notas, asignaturas, horarios, matrícula de estudiantes, solicitud de comedores, proceso de admisión, módulos financieros, entre otros, los cuales poseen grupos específicos de usuarios y que en determinadas épocas del año su demanda es grande.

Por esto es indispensable mantener en estas épocas críticas de demanda del sistema, un servidor WEB confiable y capaz de responder en forma efectiva y rápida todas las peticiones que llegan a él. Es aquí donde se encuentra la utilidad de los agentes inteligentes, ya que es muy tedioso y costoso mantener a una persona pendiente del sistema para que éste se encuentre disponible.

No obstante, es importante resaltar que el uso de agentes inteligentes no significa la no presencia de personal calificado para realizar mantenimiento del sistema. La conveniencia se encuentra en la ayuda que brindan a determinados procesos que puedan optimizar y mantener funcional el sistema.

3. OBJETIVOS

3.1 OBJETIVO GENERAL

Diseñar un modelo de agente inteligente semi-autónomo enfocado a la administración de servidores WEB como soporte al mantenimiento y ejecución de los procesos ejecutables.

3.2 OBJETIVOS ESPECIFICOS

- ✓ Analizar los conceptos básicos de los agentes inteligentes y los servidores WEB.
- ✓ Identificar los diferentes tipos de agentes.
- ✓ Analizar la utilidad de los tipos de agentes existentes para seleccionar el más adecuado para la administración de servidores WEB.
- ✓ Estudiar el uso de ontologías para la creación del comportamiento de los agentes.
- ✓ Identificar las características fundamentales que debe tener en cuenta un agente inteligente para administrar servidores WEB.
- ✓ Identificar el software existente para desarrollar agentes inteligentes.
- ✓ Plantear un modelo de agente inteligente que ejecute uno de los servicios del servidor WEB de la UIS.

4. ESTADO DEL ARTE

Desde el surgimiento del World Wide Web en la década de los 90, el cual afianzó a nivel mundial el uso del Internet, se ha visto un crecimiento exponencial de los usuarios que hacen uso de la red, sin mencionar la gran cantidad de información almacenada en miles de servidores.

Como el flujo de la información es uno de los aspectos a tener en cuenta en el momento de implementar servicios Web, la tecnología ha aportado importantes avances en cuanto al hardware de los servidores, encontrándose procesadores de alto rendimiento, bus de datos de gran capacidad, memoria de rápido acceso, entre otros.

También es importante resaltar el avance técnico y científico de los lenguajes de programación, los sistemas operativos y todos los programas implicados en el procesamiento de los datos en la red. Es aquí donde cobra vital importancia el uso de sistemas capaces de tomar las decisiones correctas en situaciones críticas en la red y en este caso de servidores Web.

Las investigaciones están enfocadas en diversas ramas. Se encuentra la Web semántica, que mediante el uso de ontologías está dando lugar al desarrollo de editores de Metadatos o editores Ontológicos como Protege o Webonto, y a sistemas para favorecer la interoperabilidad. También se trabaja en la generación de conocimiento sobre conocimientos ya especificados.[3]

Existen agentes de búsqueda inteligentes como mecanismos de recuperación de información, basados en programas que viajan por Internet buscando información que concuerde con los parámetros definidos por el usuario y evitando la sobrecarga de los servidores y contenido duplicado.[4]

[3] Tomado de http://cidlisuis.org/aedo/RGTIN2V1/RGTI_02.pdf
[4] Tomado de http://trevinca.ei.uvigo.es/~pcuesta/sm/alumnos2003/SpidersyAgentesdeBusqueda.pdf

También se encuentran agentes encargados de la indexación de bases de datos, sobre todo en los sitios Web denominados buscadores. Las empresas que ofrecen el motor de búsqueda se dedican a ejecutar un agente o varios agentes para que les vayan enviando información, que luego será indexada en la base de datos.[5]

Otras utilidades de los agentes son la validación de páginas HTML, mantenimiento de enlaces[6], monitorización de información nueva, etc.

Todos estos servicios son posibles gracias a los lenguajes de definición de ontologías, mediante el cual se define el comportamiento de los agentes. Por ejemplo OWL, lenguaje de ontologías de propósito general definido para la Web semántica.

Se puede concluir que el uso de agentes inteligentes para el manejo y administración de los datos en la red es algo en lo que se está trabajando y en un futuro será la prioridad número uno en cuanto a desarrollo de software se refiere.

[5] Tomado de http://www.lcc.uma.es/~eat/services/robots.html
[6] Uno de los agentes encargados de esta tarea se llama MOMSpider

5. SERVIDORES WEB

5.1 INTRODUCCIÓN

Para entender el concepto de servidor Web es indispensable entender la definición de Servidor, término que hoy en día es utilizado por un gran número de programadores y usuarios, pero que generalmente es asociado o aún más, conceptualizado, con una máquina que posee un hardware lo suficientemente avanzado para responder a todas las solicitudes que recibe. Por supuesto, esta definición se acerca un poco a la definición real de *Servidor*, pero olvida la verdadera esencia que abarca este concepto.

Básicamente un servidor es un software especializado que provee servicios a los usuarios o terminales que lo solicitan. Estos servicios pueden ser peticiones enviadas por un usuario conectado a su PC desde una red cualquiera o bien procesos que un PC solicite a otro sin necesidad de la intervención del usuario. Dichas peticiones van desde simples consultas hasta transacciones complejas que requieran el uso de otros servidores para su respuesta.

Por ejemplo, un cliente podría ser el estudiante que ingresa al sistema bibliográfico de la Universidad desde un PC a través de un programa de conexión de terminal segura (Putty u otro), y el servidor es quien le entrega, en formato de texto, los datos de los libros que coinciden con los parámetros de búsqueda utilizados, en respuesta a su solicitud.

Es importante resaltar que la palabra *Servidor* identifica tanto al programa como a la máquina (Hardware) en la que dicho programa se esté ejecutando.

5.2 DEFINICIÓN

Cuando la función de un computador es la de proporcionar datos en forma de páginas HTML (HyperText Markup Language) se dice que éste es un servidor Web. Dicho servicio es la respuesta de las diferentes peticiones que realizan los usuarios a través

de los navegadores[7]. Para tal efecto dicha máquina está dotada de un software que retorna datos en hipermedia, páginas HTML, textos complejos con enlaces, figuras, sonido, video, etc., es decir, todo lo que se pueda encontrar en una página de Internet.

De lo anterior se puede deducir que cualquier computador con el software adecuado y una dirección de Internet fija puede actuar como Servidor Web. El inconveniente surge cuando el número de peticiones aumente y supere la capacidad de procesamiento, almacenamiento y memoria del servidor. Es por esta razón que existen Servidores Web especializados poseedores de un hardware y un software lo suficientemente robusto para soportar la carga de solicitudes recibidas y responder a éstas en el menor tiempo posible.

Esta comunicación, entre cliente y servidor, se realiza por medio de un protocolo[8], que en el caso concreto para peticiones Web, es el protocolo HTTP[9]. Por lo anterior, un servidor Web siempre se encuentra a la espera de peticiones HTTP, la cuales son ejecutadas por un cliente HTTP, es decir, un navegador o browser.

---

[7] Aplicación que permite al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores Web de todo el mundo a través de Internet. Permiten mostrar y/o ejecutar: gráficos, secuencias de vídeo, sonido, animaciones, hipervínculos o enlaces, etc. Ejemplo de estos son el Internet Explorer, Mozilla Firefox, Netscape Navigator, Opera, entre otros.

[8] Conjunto de reglas que gobiernan el intercambio de datos entre entidades. Se podría también definir como el lenguaje que utilizan los computadores para comunicarse y entenderse entre sí. Algunos ejemplos de protocolos son: FTP, POP3, SMTP, ICMP, etc.

[9] HTTP es un protocolo que se caracteriza porque no es permanente, es decir, cada operación HTTP implica una conexión con el servidor, que es liberada al término de la misma. Es decir, un documento HTML con 7 imágenes requiere 8 conexiones distintas (7 imágenes más la página HTML en sí). Además cada petición es tratada como una operación totalmente independiente de las demás.

A partir de la versión 1.1 del protocolo HTTP, se pueden habilitar conexiones persistentes, las cuales permiten enviar más objetos con menos conexiones.

5.3 FUNCIONAMIENTO

El esquema de funcionamiento de un servidor Web se basa en la ejecución infinita del siguiente bucle (ver figura 1) [1]:

1. Esperar peticiones en el puerto TCP indicado que por defecto para HTTP es el 80.
2. Recibir una petición.
3. Buscar el recurso.
4. Enviar el recurso utilizando la misma conexión por la que recibió la petición.
5. Volver al punto 2.

El servidor es el responsable de proporcionar el acceso a los recursos solicitados que están bajo el control del sistema operativo. Estos recursos pueden ser estáticos, como páginas HTML o texto y dinámicos, como por ejemplo JSP's[10], los cuales son ejecutados por el servidor. La siguiente figura muestra la interacción del servidor Web y su entorno:

[10] Java Server Pages: Tecnología JAVA que permite a los programadores generar contenido dinámico en forma de documentos HTML, XML, o de otro tipo.

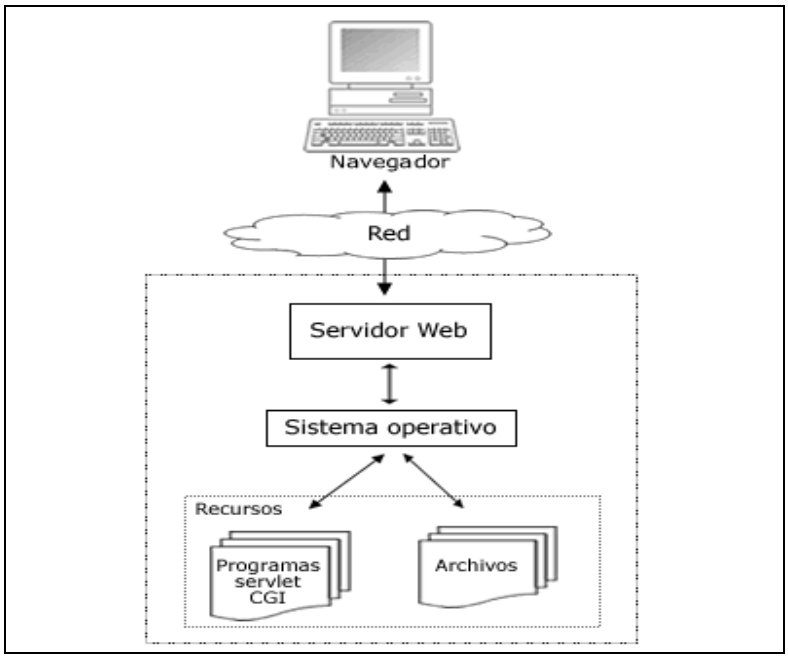

Figura 1. Interacción de un servidor con su entorno

5.4 ARQUITECTURA

La arquitectura de un servidor Web está dividida en dos capas, *la capa servidor* y *la capa soporte (Ver figura 2) [2].*

La capa servidor está compuesta por cinco subcapas, responsables de implementar la funcionalidad de un servidor Web.

- ✓ *Subcapa de recepción:* Su función es la de esperar y analizar las peticiones HTTP de los clientes y determinar las características de los navegadores (tipo de navegador, compatibilidad, entre otras). Esta subcapa contiene la lógica necesaria para manejar múltiples peticiones.

- ✓ *Analizador de peticiones:* Es el encargado de traducir la localización del recurso de la red al nombre del archivo local. Por ejemplo, la solicitud del recurso

http://www.uis.edu.co se traduce al archivo local /usr/local/jakarta-tomcat-5.0.28/webppas/portal/index.jsp.

- ✓ *Control de acceso:* Encargado de la autenticación y acceso al sistema.

- ✓ *Manejador de recursos:* Esta subcapa es la responsable de determinar, ejecutar y generar la respuesta del tipo de recurso solicitado.

- ✓ *Registro de transacción*: Encargado de registrar todas las peticiones y su resultado.

La capa de soporte actúa como una interfase entre el sistema operativo y el servidor Web y, entre las propios subcapas de la capa superior. Esta compuesta por las siguientes subcapas:

- ✓ *Util:* Contiene funciones que son utilizadas por el resto de las subcapas.

- ✓ *Capa abstracta del Sistema Operativo (OSAL):* Esta subcapa encapsula el funcionamiento específico del sistema operativo para facilitar la portabilidad del servidor Web a diferentes plataformas.

La siguiente figura muestra la arquitectura expuesta anteriormente:

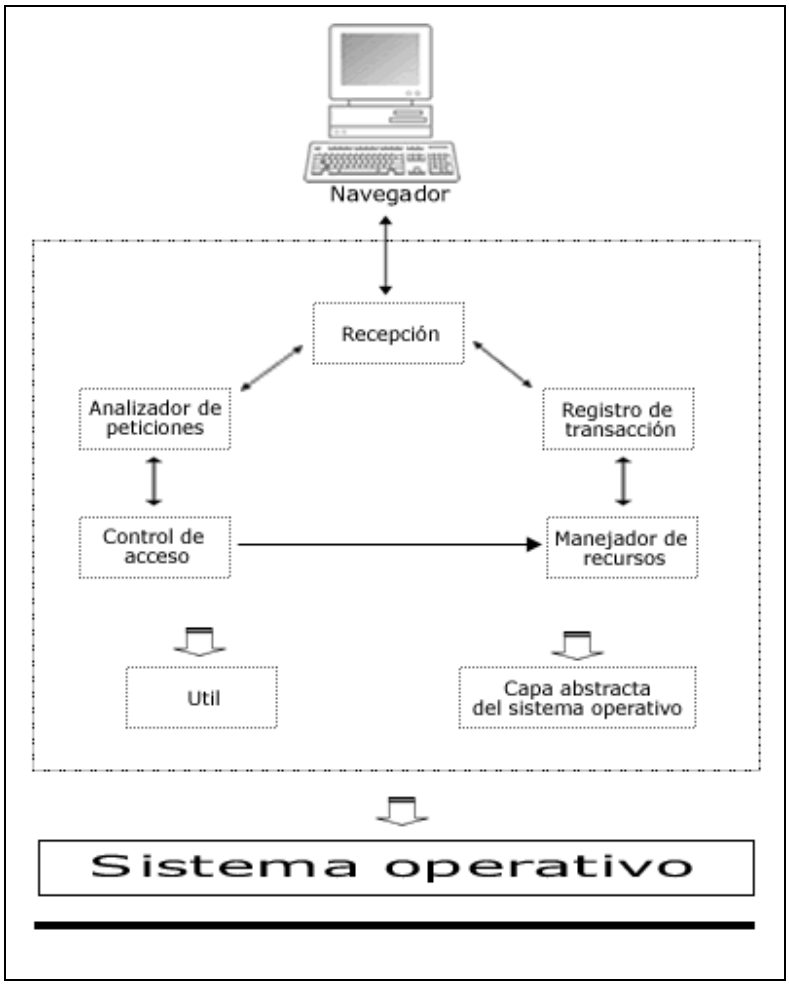

Figura 2. Arquitectura de un servidor Web

5.5 CARACTERISTICAS

Todos los servidores Web deben incluir la capacidad de acceder a contenido estático o archivos que se encuentren en alguna parte del disco. Esta parte debe estar especificada en la configuración del servidor y puede ser variable dependiendo de las necesidades del sistema, es decir, se pueden configurar varios directorios de acceso en donde el servidor buscará los archivos solicitados de acuerdo al tipo de solicitud.

Algunos servidores Web permiten también especificar directivas de seguridad por directorio mientras que otros hacen posible la especificación de los archivos que se deben considerar como índice del directorio.

Igualmente, otro aspecto fundamental en un servidor Web es el soporte que ofrece para acceder y enviar contenido dinámico a los usuarios. La mayor parte de los servidores Web ofrecen soporte para CGI[11]. Otros ofrecen soporte para algunos lenguajes de programación como PHP[12], JSP, ASP[13], lo cual es muy recomendable a la hora de escoger el servidor.

También tienen la capacidad de relacionar múltiples dominios[14] a través de una única dirección IP mediante servidores virtuales. Esto permite la administración racional de las direcciones IP y una configuración distinta para cada servidor.

5.6 TIPOS DE SERVIDORES

A continuación se mencionan los tipos de servidores Web que existen [2][6]:

5.6.1 Servidores basados en procesos

Se consideran los predecesores de todos los demás tipos de servidores. Se basan en la obtención de paralelismo mediante la duplicación del proceso[15] de ejecución.

Sobre éste existen varios modelos. El más sencillo consiste en generar una copia exacta de la conexión entrante para que sea la encargada de atender la conexión.

[11] Common Gateway Interface o Pasarela de Interface Común. Es un protocolo de comunicación entre un servidor Web y una aplicación externa, es decir, el servidor utilizando el protocolo CGI, ejecuta un programa especificado por la página Web, seguidamente enviando los datos solicitados al servidor (a través de la interface CGI), y a su vez éste retorna los retorna al cliente en forma de página HTML.

[12] PHP es un lenguaje de programación cuyas siglas significan PHP Hypertext Pre-processor.

[13] ASP es una tecnología del lado servidor de Microsoft para páginas Web generadas dinámicamente, que ha sido comercializada como un anexo a Internet Information Server (IIS).

[14] Nombre único que identifica a un sitio Web en Internet

[15] Ocurrencia o instancia de un programa en ejecución, el cual es propietario de una serie de recursos como: espacio de direcciones en memoria, archivos, etc.

Sobre este modelo se generaron optimizaciones importantes, como las que incluyó Apache con la técnica de pre-fork, en la cual un proceso de control es el responsable de crear los procesos hijo encargados de “escuchar” y atender las peticiones que se puedan producir. En este caso especial, Apache mantiene varios procesos disponibles de más, para ser utilizados en otras peticiones. Así, los usuarios no tienen que esperar a que un nuevo proceso hijo sea creado para ser atendidos. En conclusión la técnica pre-fork esta fundamentada en una colección de conexiones.

Las principales ventajas de este diseño residen en su simplicidad de implementación y su seguridad. Su desventaja radica en el bajo rendimiento. La creación o eliminación de un proceso son tareas pesadas para el sistema operativo y consumen una gran cantidad de tiempo.

5.6.2 Servidores basados en hilos

Poseen características de los servidores basados en procesos pero sustentadas en el concepto de hilo[16].

Su ventaja radica en el bajo costo para la creación de un hilo comparada con la de un proceso. Además múltiples hilos pertenecientes a un mismo proceso pueden compartir datos entre ellos, ya que comparten el mismo espacio de memoria. Además, si no hay hilos ejecutando código en el espacio de direcciones del proceso, el sistema destruirá automáticamente el proceso y su espacio en memoria.

El modelo de servidor basado en hilos hereda muchas de las características de los servidores basados en procesos, entre ellas la de la simplicidad en su diseño e implementación. Por otro lado, el compartir el espacio de memoria implica un riesgo de seguridad que no tienen los servidores basado en procesos.

Este tipo de servidor, hoy en día, es mucho más común que el basado en procesos.

[16] Los hilos son una forma de dividir un programa en dos o más tareas que corren simultáneamente.

5.6.3 Servidores basados en *sockets* no bloqueantes o dirigidos por eventos

Estos servidores basan su funcionamiento en la utilización de lecturas y escrituras asíncronas sobre sockets[17]. Normalmente, estos servidores utilizan una llamada al sistema para examinar el estado de los sockets con los que trabaja. Cada sistema operativo implementa una o más funciones de examen de sockets. El objetivo de estas funciones es inspeccionar el estado de un grupo de sockets asociados a cada una de las conexiones.

La ventaja de este diseño es principalmente su velocidad y su desventaja es que la concurrencia es simulada, es decir, existe un sólo proceso y un sólo hilo, desde el cual se atienden todas las conexiones.

Actualmente existen una serie de servidores Web de libre distribución basados en este diseño de servidor, de los cuales se destacan *thttpd* y *Boa*.

5.6.4 Servidores basados en el Kernel

No poseen un diseño definido. Es un intento de acelerar la velocidad de un servidor Web mediante el movimiento de su código de espacio de usuario a espacio de kernel, lo que permite optimizar las operaciones de interacción directa con la pila TCP/IP.

Por otro lado, no es necesario realizar el intercambio de información entre el espacio del kernel y el del usuario como ocurre en otros servidores. De esta manera al realizar la misma tarea la implementación en el kernel resulta ser más eficiente.

[17] Puntos o medios de comunicación entre dos aplicaciones que permiten que un proceso emita o reciba información con otro proceso estando los dos en distintas máquinas.

Aparentemente este modelo es muy eficiente, pero en la práctica los inconvenientes son muy grandes, debido a que cualquier problema que se produzca a nivel de kernel puede ocasionar la caída de todo el sistema y por ende la caída de todos los programas que se estén ejecutando.

Ejemplo de servidores empotrados en el kernel: *Tux* y *Afpa*

5.7 EJEMPLOS DE SERVIDORES

Algunos ejemplos de servidores Web que actualmente funcionan en Internet son los siguientes:

5.7.1 Servidor Apache

El servidor HTTP Apache es un servidor HTTP de código abierto para plataformas Unix, Windows y otras, que implementa el protocolo HTTP/1.1 (RFC 2616 [9]) y la noción de sitio virtual. Puede funcionar como un proceso standalone, sin que eso solicite el apoyo de otras aplicaciones o directamente del usuario [5][7].

El servidor Apache se desarrolla dentro del proyecto HTTP Server (httpd) de la Apache Software Foundation. Al comienzo de su desarrollo, en 1995, se basó inicialmente en código del popular NCSA[18] HTTPd 1.3, y más tarde fue reescrito por completo, solucionando los problemas existentes, manteniendo algunas características e incluyendo otras nuevas.

Su nombre se debe a que originalmente consistía solamente en un conjunto de parches que se aplicaban al servidor de NCSA (patchy server - un servidor parcheado).

Algunas de sus características son:

[18] National Center for Supercomputing Applications

- ✓ *Soporte DSO (Dynamic shared objects):* Los módulos se pueden cargar si se solicita, para utilizar una menor cantidad de memoria. Esta característica se desarrolla en: FreeBSD, OpenBSD,NetBSD, Linux, Solaris, SunOS, Digital UNIX, IRIX, HP/UX, UnixWare,AIX, ReliantUnix y las demás plataformas SVR4.

- ✓ soporta tanto los virtualhosts que se basan en el número de IP como los que se basan en el nombre

- ✓ Gestión de proxy server.
- ✓ Mensajes de error altamente configurables.
- ✓ Bases de datos de autenticación.
- ✓ Soporta contenido dinámico.
- ✓ Permite utilizar el protocolo HTTP con el soporte SSL (HTTPS).

5.7.2 Internet Information Services (IIS)

Servicios de software que admiten la creación, configuración y administración de sitios Web, además de otras funciones de Internet. Entre los servicios se encuentran FTP[19], SMTP[20], NNTP[21] Y HTTP/HTTPS.

Originalmente era parte del Option Pack para Windows NT. Luego fue integrado en otros sistemas operativos de Microsoft destinados a ofrecer servicios, como Windows 2000 o Windows Server 2003.

---

[19] File Transfer Protocol (Protocolo de Transferencia de Archivos), utilizado para transferir grandes bloques de datos por la red. Su comportamiento está definido por la recomendación RFC 959.

[20] Simple Mail Transfer Protocol (SMTP), o protocolo simple de transferencia de correo electrónico. Protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre computadoras o distintos dispositivos.

[21] Network News Transport Protocol (NNTP), o protocolo de transferencia de noticias. Es el Protocolo de red utilizado por el Usenet internet service. Es un Protocolo de red basado en tiras de textos enviados sobre canales TCP de 7 bit ASCII . Es usado para subir y bajar así como para transferir artículos entre servidores.

Este servidor Web se basa en módulos que dan capacidad para procesar distintos tipos de páginas. Microsoft incluye los Active Server Pages (ASP) y ASP.NET. También pueden ser incluidos los de otros fabricantes, como PHP o Perl.

Algunas versiones son:

- ✓ IIS 1.0, Windows NT 3.51 Service Pack 3
- ✓ IIS 2.0, Windows NT 4.0
- ✓ IIS 3.0, Windows NT 4.0 Service Pack 3
- ✓ IIS 4.0, Windows NT 4.0 Option Pack
- ✓ IIS 5.0, Windows 2000
- ✓ IIS 5.1, Windows XP Professional
- ✓ IIS 6.0, Windows Server 2003 y Windows XP Professional x64 Edition
- ✓ IIS 7.0, Windows Vista y Windows Server "Longhorn"

5.7.3 Lighttpd

Lighttpd es un servidor Web diseñado para ser rápido, seguro, flexible, y fiel a los estándares. Está optimizado para entornos donde la velocidad es muy importante, y por eso consume menos CPU y memoria RAM que otros servidores. Ideal para cualquier servidor que tenga problemas de carga.

Lighttpd es software libre y se distribuye bajo la licencia BSD[22]. Funciona en GNU/Linux y otros sistemas operativos tipo UNIX.

Algunas de sus características son:

- ✓ Virtual hosting
- ✓ Soporte para PHP y otros

[22] La licencia BSD es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Pertenece al grupo de licencias de software Libre. Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre.

- ✓ Cifrado SSL
- ✓ Autenticación
- ✓ Consumo de memoria constante
- ✓ Redirecciones HTTP, y reescrituras de URL
- ✓ Capacidad para enviar partes de un archivo
- ✓ Realización de estadísticas
- ✓ Redirección condicional
- ✓ Permite módulos externos
- ✓ Permite comunicarse con programas externos mediante FastCGI o SCGI, que son mejoras al CGI original (también soportado). De esta forma, se pueden usar programas en prácticamente cualquier lenguaje de programación.

5.7.4 Tomcat

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) funciona como un contenedor de servlets[23] desarrollado bajo el proyecto Jakarta en la Apache Software Foundation [10].

El proyecto de Yakarta Tomcat tiene sus orígenes con la aparición de los contenedores de servlets de java. El primero fue creado por Sun Microsystems y se denominó Java Web Server, pero no fue lo suficientemente robusto. Mientras tanto, la Apache Software Foundation (ASF) creó el JServ, motor que se integró con el Apache Web Server.

En 1999, Sun Microsystems donó el código de su contenedor de servlets a la ASF y los dos proyectos se unieron para crear el servidor Tomcat.

---

[23] Los servlets son objetos que corren dentro del contexto de un servidor Web y extienden su funcionalidad. Su uso más común es la de generar páginas Web de forma dinámica a partir de los parámetros de la petición que envíe el navegador Web.

Al principio, se tenía la idea de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. En la actualidad esa percepción ha desaparecido y es usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Tomcat fue escrito en Java y es por esta razón que funciona en cualquier sistema operativo que disponga de la máquina virtual. Además implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Sun Microsystems.

La primera versión de Tomcat fue la 3.x y sirvió de referencia para las especificaciones del servelt 2.2 y JSP 1.1. La serie 3.x era descendiente del código que originalmente Sun entregó a la ASF en 1999.

En el 2001, Tomcat 4.0 (Código denominado Catalina) fue lanzado. Catalina fue un rediseño de la arquitectura Tomcat y construyó una nueva base de código. La serie 4.x utiliza el servlet 2.3 y JSP 1.2.

Tomcat 5.0 es una de las últimas versiones de Tomcat. Utiliza servelt 2.4 y JSP 2.0 además de nuevas características y optimizaciones en rendimiento.

5.7.5 Websphere Application Server

Aplicación perteneciente al grupo Websphere de IBM que permite el despliegue de aplicaciones basadas en tecnologías J2EE[24].

Posee características muy poderosas para el desarrollo y administración de aplicaciones Web complejas. Se trata de un entorno de desarrollo basado totalmente en Java. Este hecho implica que en las aplicaciones se deba utilizar Servlets para

[24] J2EE son las siglas de Java 2 Enterprise Edition que es la edición empresarial del paquete JAVA creada y distribuida por Sun Microsystems. Comprenden un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales.

generar el contenido y acceder a los datos, Enterprise JavaBeans (EJB) con el objeto de encapsular los componentes de negocio, y páginas JSP actuando como enlace entre los diferentes componentes.

Se adopta también el estándar XML para facilitar la separación entre el contenido Web y la lógica de las aplicaciones, así como para el intercambio de datos. Entre sus características se encuentran significativas mejoras en cuanto a escalabilidad y rendimiento en los componentes Java y páginas JSP, un soporte más optimizado de transacciones y persistencia, unos mayores controles de seguridad y facilidad de administración, y una potente herramienta para registro, análisis y estadísticas del uso de las aplicaciones.

Sus ventajas radican en toda la gama de funciones que ofrece, pero su desventaja se encuentra en el hecho de que no es un servidor libre.

5.7.6 Thttpd

Este servidor forma parte de un conjunto de servidores existentes denominados servidores ligeros, formado entre otros por thttpd, boa, LiteSpeed y LightHttpd. Su diferencia con otros como Apache o IIS, radica en su baja potencialidad y flexibilidad, pero mucho más rápidos aprovechando al máximo los recursos disponibles.

Su principal utilidad se basa en servir páginas estáticas como imágenes, en las cuales no hace falta toda la flexibilidad que aportan servidores más grandes como Apache.

6. AGENTES INTELIGENTES

Actualmente existe mucho interés en la tecnología de agentes inteligentes. Esto se debe a que el desarrollo de software se está orientando a la creación de programas y sistemas complejos que puedan cubrir las necesidades y requerimientos de los usuarios finales, desde simples recomendaciones en el uso de herramientas hasta la solución acertada de un sinnúmero de problemas.

Para comprender el concepto de agente inteligente es importante tener claro los conceptos de *agente* e *inteligencia*.

La palabra *agente* proviene del latín agere que significa "el que hace". Se puede entonces afirmar que un agente es todo aquello que percibe su entorno mediante sensores y cuya respuesta a éste lo realiza mediante efectores [13]. La siguiente figura resume este concepto:

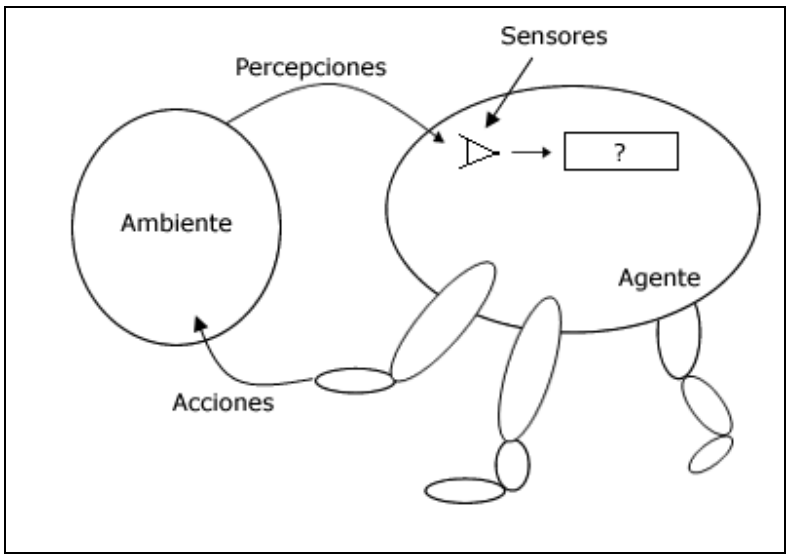

Figura 3. Agente inteligente

Esta definición, al igual que la estructura de los agentes, esta basada en la organización anatómica de los seres humanos. Los ojos, oídos y demás sentidos son los sensores. Las manos, piernas y otras partes del cuerpo son los efectores.

Es por ello que cualquier desarrollo de agentes se asemeja a esta estructura. Por ejemplo, en los agentes robóticos los sensores son las cámaras infrarrojas y los efectores son los motores.

Desde el punto de vista computacional se puede decir que un agente software es aquel que se ejecuta en un ambiente en donde realizan ciertas tareas con el objetivo de cumplir las metas para las cuales fue programado.

La *inteligencia* es una facultad especial propia de los seres humanos que le otorga la capacidad de tomar decisiones en su obrar. Cuando la inteligencia se usa con la conciencia dota al ser vivo de voluntad, de la existencia e individualidad, así como también de los medios de establecer relaciones con el mundo exterior y atender a sus necesidades [35].

Por lo anterior se puede definir un agente inteligente como un sistema de computación que sustituye a una persona o proceso para llevar a cabo una actividad o para cumplir con un objetivo. Esta entidad ofrece capacidades para tomar decisiones similares a las tomadas por los seres humanos.

Además de ésta se pueden encontrar otras definiciones, implícitas en la expuesta anteriormente [16]:

Los agentes inteligentes son programas que realizan tareas interactivas, dirigidas por la petición y los deseos de los usuarios. Poseen un grado de autonomía e independencia, en virtud del cual pueden realizar una serie de tareas sin que las personas u otros agentes les dirijan en cada paso que dan en su camino.

Según Sankar Virdhagriswaran de Crystaliz Inc., y basado en el agente móvil MuBot:

El término agente se usa para representar dos conceptos ortogonales. El primero es la habilidad de un agente para la ejecución autónoma. El segundo es la habilidad que tiene un agente para hacer razonamientos orientados al dominio que se esté tratando.

La siguiente tabla resume los elementos o propiedades que pueden caracterizar a un agente:

Tabla 1. Propiedades de los agentes inteligentes

Propiedad	Descripción
Inteligencia	Es el grado de razonamiento y capacidad de aprendizaje que posee el agente para conseguir su objetivo. Varía de unos agentes a otros y dependen de la tecnología implementada en sus módulos. Dicha tecnología se fundamentada en la Inteligencia Artificial. La inteligencia implementada va desde la simple captura de la configuración del usuario para realizar una tarea, hasta la tecnología de redes neuronales especializada en donde el agente es capaz de modificar su configuración para optimizar ciertas actividades e ir ampliando su grado de conocimiento para la ejecución de éstas.
Autonomía	Es la capacidad que tiene un agente de actuar sin necesidad de intervención humana o de otros agentes, pero con un pequeño grado de control sobre sus acciones. Es una de las principales características de los agentes, ya que en ésta se basan los agentes que modifican su conducta de acuerdo al grado de aprendizaje que han adquirido a través de su experiencia.
Comunicación	Es la capacidad que tiene el agente para comunicarse con el usuario, con otros agentes y con otros programas.

	La comunicación con el usuario se realiza mediante interfaces, las cuales pueden estar en forma visual o auditiva[25]. En cualquier caso, éstas deben ser amigables y permitir modificar la configuración del agente.
Sensibilidad	Capacidad para interactuar con el usuario u otros agentes utilizando un lenguaje en particular. En el caso del usuario, un lenguaje que éste pueda entender fácilmente. Y en el caso de otros agentes o programas mediante la utilización de Ontologías[26].
Reactividad	Capacidad para responder a los cambios en el entorno con una acción especifica que le permita continuar ejecutando la tarea que se estaba realizando o simplemente optimizarla.
Proactividad	Capacidad para presentar un comportamiento particular en base a las metas programadas. En este caso se ven reflejadas las decisiones tomadas por el agente en las acciones tomadas para alcanzar su objetivo.
Continuidad	Esta característica hace referencia al trabajo constante realizado por el agente para percibir sucesos del medio que le permitan realizar acciones. Son los procesos que se procesan constantemente por el sistema y que se ven reflejados en la ejecución del agente.
Colaboración	En la interacción con los usuarios, los agentes pueden solicitar acciones de éstos para mejorar los procesos ejecutados para que se realicen de forma más eficaz y eficiente. Así mismo pueden sugerir al usuario que realice ciertas tareas para optimizar su configuración.
Benevolencia	El agente es capaz de satisfacer todas las solicitudes del usuario.

[25] Algunos agentes permiten comunicarse en lenguaje natural. Por ejemplo los chatbots.
[26] El término se amplía mas adelante en el capítulo dedicado a Ontologías.

6.1 CLASIFICACIÓN DE LOS AGENTES

Russell and Norvig clasifican los agentes inteligentes de acuerdo a las acciones que realizan. A continuación se presenta un resumen de esta clasificación haciendo énfasis en el tipo de agente elegido para el desarrollo de este proyecto.

6.1.1 Agentes basados en la relación percepción – acción

Esta clasificación se rige por la forma en que un agente reacciona ante un evento del medio en el que se ejecuta. Esta clasificación está fundamentada en un conjunto de reglas codificadas que soportan las decisiones y el comportamiento que se toman en determinados momentos.

Basados en lo anterior, el agente a desarrollar se acomoda perfectamente dentro de esta clasificación, ya que necesita, de acuerdo a unas reglas, tener una percepción de lo que está pasando con el Servidor Web y ejecutar ciertas acciones que le permitan restablecer el sistema si fuera el caso.

Dentro de esta clasificación se encuentran establecidos los siguientes tipos de agentes [14]:

- ✓ **Agentes reflejo simple**

Se basa en el paradigma causa – efecto, que para el caso de los agentes se denomina condición – acción.

De acuerdo con esta regla se establece la conexión entre las percepciones y las acciones. El agente consulta la regla cuya condición coincida con la situación definida por una percepción y ejecuta la acción correspondiente. En la siguiente figura se aprecia su estructura, en donde el óvalo es el conocimiento y el rectángulo el estado interno.

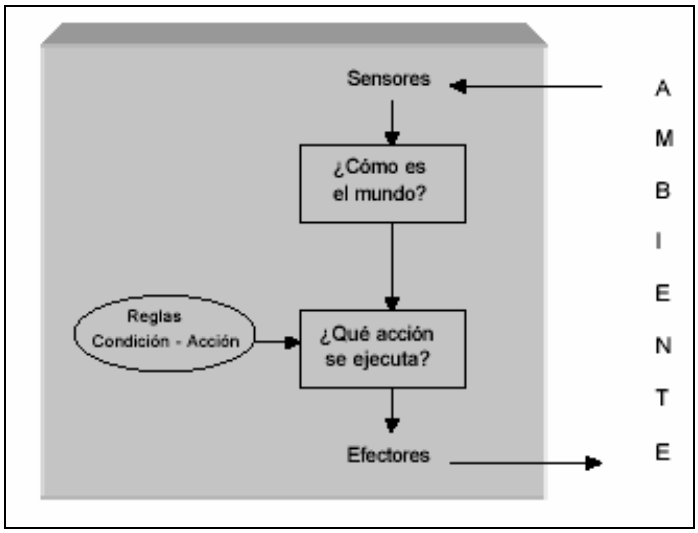

Figura 4. Agente reflejo simple

Este agente funciona correctamente mientras se tome la decisión adecuada en un instante determinado. Además es fácil de implantar con la limitante de que el ámbito de la aplicación es bastante reducido.

✓ **Agentes bien informados de todo lo que pasa**

Como se mencionó anteriormente, los agentes reflejo simple ejecutan acciones en un instante dado de acuerdo a las reglas programadas. Pero en ciertos casos es posible que el agente no solucione el problema ya que existen situaciones en las que un evento no puede asegurar una acción que permita obtener el resultado más óptimo. Es en estos casos en los que es necesario analizar las percepciones anteriores y de esta manera ejecutar la acción mas adecuada.

Por lo anterior, el agente debe estar dotado de dos tipos de conocimiento: el que está relacionado con la evolución del entorno y el relacionado con el efecto que tienen las acciones al ejecutarse en el medio. Esto se puede apreciar en la siguiente figura:

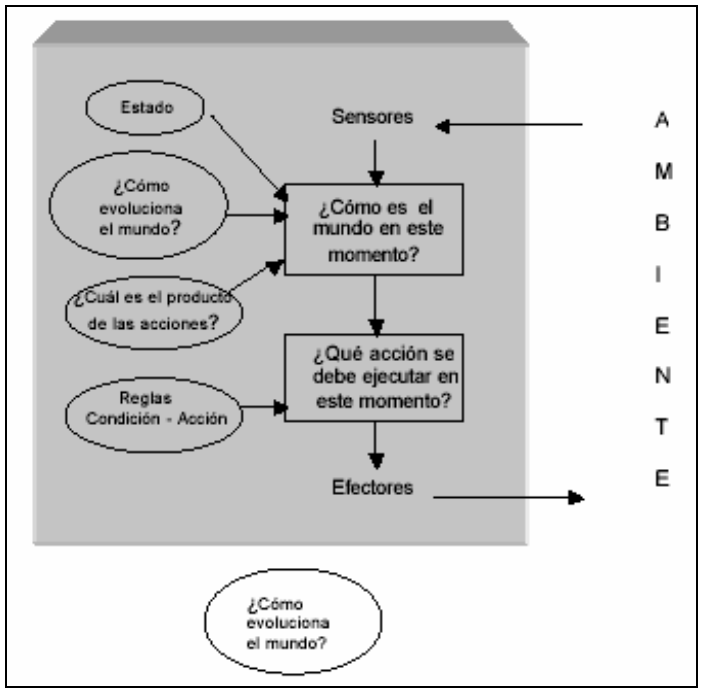

Figura 5. Agente bien informado de lo que pasa

✓ **Agentes basados en metas**

En el tipo de agente expuesto anteriormente, se puede apreciar claramente que se toman decisiones sustentadas en eventos anteriores, pero en ningún momento el agente se toma la molestia de consultar si dichas acciones le permiten alcanzar el objetivo de una manera óptima.

Por ello, para decidir qué acción realizar en un momento determinado no basta con tener información referente al estado que prevalece en el ambiente sino que se necesita algo adicional que represente las metas o las situaciones deseables. Es decir, el agente ahora debe preguntarse que sucederá si se toma X o Y decisión. Para ello se deben utilizar técnicas de Inteligencia Artificial de búsqueda y planificación.

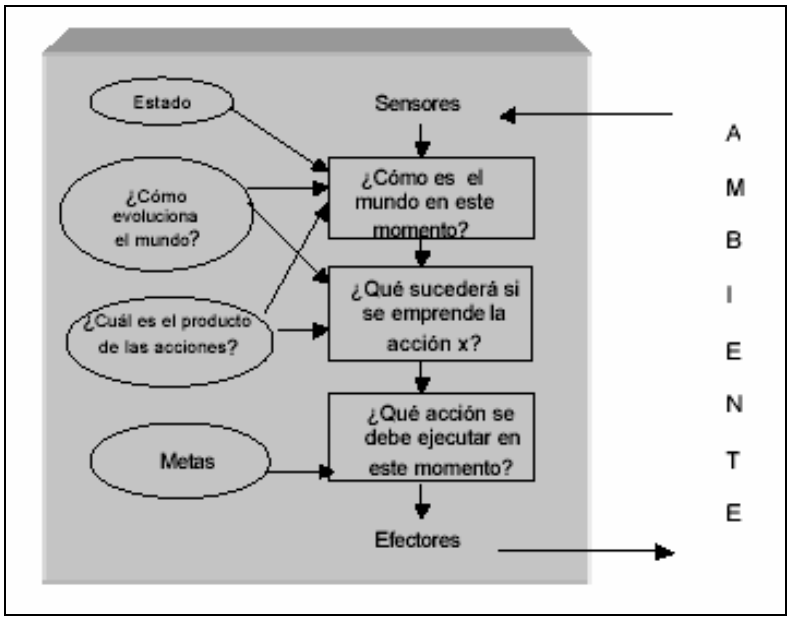

Figura 6. Agente basado en metas

✓ **Agentes basados en utilidad**

La utilidad es una función que correlaciona un estado o secuencia de estados (cuando se mide la utilidad de un agente a largo plazo) y un número real mediante el cual se caracteriza el correspondiente grado de satisfacción [11].

Las metas permiten establecer una diferencia entre los estados deseables o indeseables, mientras que mediante una medida de desempeño más general, sería posible establecer una comparación entre los diversos estados del mundo o secuencias de estados.

La completa especificación de la función de utilidad permite la toma de decisiones racionales en dos tipos de casos en los que las metas se encuentran con problemas:

✓ Cuando el logro de alguna meta implica un conflicto y sólo algunas de ellas se pueden obtener, la función de utilidad definirá cuál es el compromiso adecuado por el que hay que optar.

- ✓ Cuando son varias las metas que el agente podría desear obtener, pero no existe la certeza de poder lograr ninguna de ellas, entonces la utilidad es una vía para ponderar la posibilidad de tener éxito considerando la importancia de las diferentes metas.

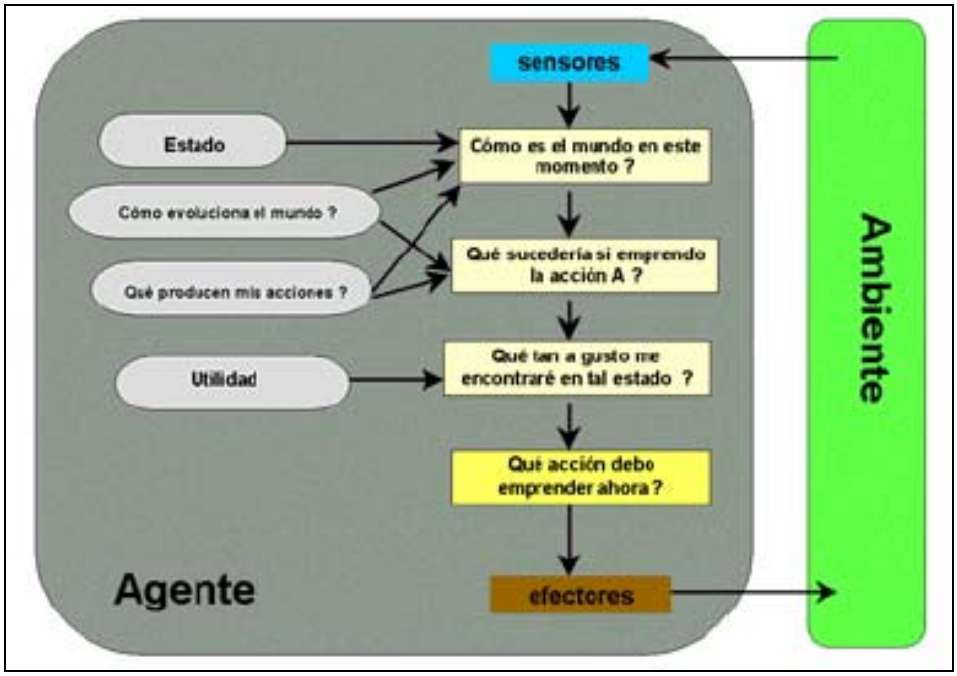

Figura 7. Agente basado en utilidad

De acuerdo a los tipos de agentes que abarca esta clasificación, se puede identificar claramente que el agente a desarrollar es un agente reflejo simple: el agente consulta el sistema de acuerdo a las reglas establecidas, identifica la acción relacionada con ésta y la ejecuta. No obstante es importante resaltar que se puede implementar un agente basado en metas, pero para ello se requiere la investigación del servidor a nivel de programación y teniendo como base el agente reflejo simple.

6.1.2 Agentes basados en el tipo de aplicación

✓ **Agentes de interfaz**

Se utilizan en sistemas complejos en donde la información que se proporciona al usuario es grande y, en muchas ocasiones, poco entendible. Por esta razón, la principal característica de este tipo de agente es la de hacer comprensibles las interfaces.

✓ **Agentes de búsqueda de información o rastreador**

Encargados de rastrear en las redes la información solicitada. Pueden ser configurados por el usuario, además aprenden de sus búsquedas. Cuando rastrean la información solicitada en la red informan por correo electrónico las novedades que consideran pueden ser de interés para el usuario.

Algunos ejemplos de este tipo de agente son [13]:

- *Maimai*[27] agiliza la búsqueda de los anuncios clasificados tras haber estudiado el comportamiento de sus visitantes durante varios meses. De esta forma, si un navegante está buscando un Mazda 6 y en ese momento no hay ningún modelo disponible, el agente de inteligencia relaciona rápidamente este vehículo con otros de características similares, y así ofrece alternativas razonables. Por otra parte, este programa informático ofrece la posibilidad de avisar al interesado, mediante un sistema de alerta al correo electrónico o al teléfono móvil, cuando disponga de alguna oferta que hubiera sido solicitada previamente.

- *Google News*[28]. Permite al usuario crear alertas de forma que avisen cuando exista una noticia sobre la palabra solicitada.

[27] http://maimai.com
[28] http://news.google.com

- ✓ **Agentes secretos o espías**

Su objetivo es monitorizar una página Web identificada previamente por el usuario. Por ejemplo una página con la información financiera de una empresa, una con ofertas de empleo de una empresa, etc. e informan cuando se producen cambios en dicha página.

Algunos ejemplos de este tipo de agente son Spyweb[29] y Changedetection[30]

- ✓ **Agentes concejeros**

También llamados asistentes personales. Su objetivo es la de ayudar al usuario como lo hace una ayudante. También son utilizados en sistemas de diagnóstico.

- ✓ **Agentes de recuperación de información**

Especializados en la búsqueda y recuperación de información en grandes bases de datos, de conocimiento o de documentos.

- ✓ **Agentes de sistemas**

Realizan inventario de hardware, eventos de red, detectar virus, etc. especialmente en sistemas distribuidos.

- ✓ **Agentes negociadores en mercados electrónicos**

Localiza una subasta en Internet, aprende cómo va la puja y realiza la compra por el usuario. También inspecciona en las tiendas más baratas. Se pueden encontrar ejemplos en http://www.botspot.com/BOTSPOT/Windows/Shopping_Bots/Auction_Bots

[29] http://www.spypress.com/spyweb.php
[30] http://www.changedetection.com

6.2 SISTEMAS MULTIAGENTE

Los sistemas multiagente son sistemas constituidos de múltiples agentes que se integran y trabajan juntos, para llevar a cabo una serie de tareas y cumplir con unos objetivos. Estos objetivos pueden ser comunes a todos o a un agente en concreto [13].

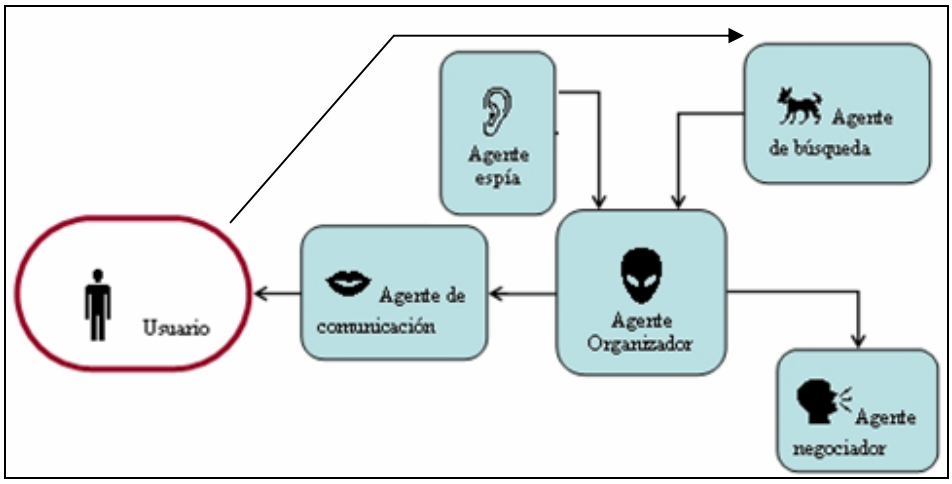

Figura 8. Ejemplo de un sistema multiagente

Los agentes pertenecientes a un sistema multiagente pueden ser heterogéneos u homogéneos, colaborativos o competitivos, etc. La definición de los tipos de agentes depende de la finalidad para la cual son construidos.

Los sistemas multiagentes compuestos por agentes reflejo simple están constituidos por un gran número de éstos. Son bastante simples, no poseen inteligencia o representación de su ambiente y se integran utilizando un comportamiento de acción – reacción. La inteligencia aparece gracias a la interacción con el medio.

Los sistemas multiagentes constituidos por agentes que poseen una base de conocimiento (los bien informados en todo lo que pasa, basado en metas y utilidad) están generalmente constituidos por una cantidad mucho menor de agentes en comparación con los compuestos por agentes reflejo simple. Estos sistemas son inteligentes y contienen una representación parcial de su medio y de los otros agentes.

Por tanto, pueden comunicarse entre si, intercambiar información o servicios y planear una acción futura. Todas estas características les permiten coordinar sus acciones en la solución de un problema.

En términos prácticos, cuando se desea realizar una tarea mediante agentes inteligentes, no se hace con uno solo de éstos, pues se requiere la ayuda de otros para poder completar las metas propuestas.

Las ventajas de trabajar con sistemas multiagente son:

- ✓ Se fortalece la robustez y la eficiencia
- ✓ Capacidad de permitir interoperabilidad entre sistemas
- ✓ Capacidad de resolver problemas, en especial en sistemas distribuidos

Pero a pesar de las ventajas que poseen, también se encuentran algunas dificultades en su implementación:

- ✓ Estilo de programación, descomposición y síntesis de resultados con un grupo de agentes.
- ✓ Protocolos de comunicación entre agentes.
- ✓ Asegurar la coherencia en la toma de decisiones.
- ✓ Mediar entre diferentes puntos de vista de acuerdo a los agentes que intervienen.

6.3 PLATAFORMA JADE (Java Agent DEvelopment framework)

JADE es una plataforma de desarrollo de aplicaciones multiagente basada en los estándares de FIPA. Ha sido implementado completamente en Java como código abierto. Se compone de dos elementos principales: una plataforma para agentes conformes a las especificaciones de FIPA y un paquete para desarrollar estos agentes en Java [33].

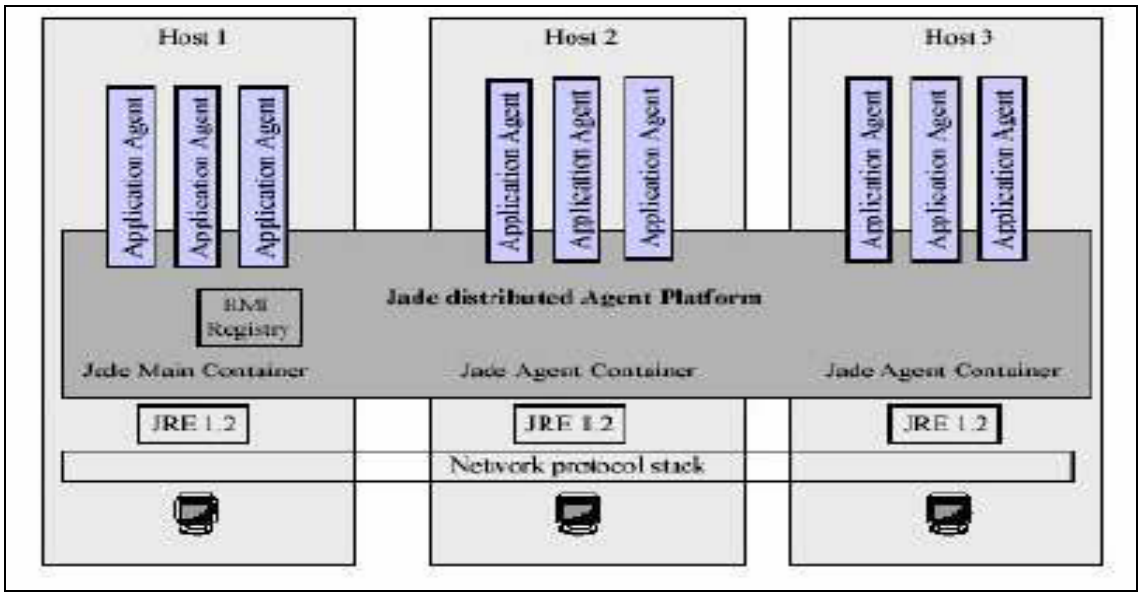

Figura 9. Plataforma JADE

JADE incluye funcionalidades para la creación básica de agentes, la programación básica del comportamiento de agentes, ACL FIPA para envío y recepción de mensajes, clases útiles para programación de protocolos, manejo de conocimiento mediante ontologías y codecs como RDF, SL, etc.

JADE Puede ejecutarse en una o varias máquinas virtuales (JVM), y cada una de éstas es concebida como un entorno en donde los agentes pueden ejecutarse concurrentemente e intercambiar mensajes.

Se encuentra organizado en contenedores (Containers), cuyo contenedor principal es AMS en donde se encuentran el DF[31] y el registro RMI[32].

En cuanto al comportamiento de agentes se puede decir que JADE propone el modelo de un único proceso con un nivel de cronometrización[33] sobre éste; lo que conlleva a una programación basada en comportamientos que determina que debe ser capaz de

[31] Agente encargado de proveer el servicio de páginas amarillas, es decir, un directorio de agentes.
[32] Remote Method Invocation
[33] Un solo comportamiento se ejecuta cada instante

hacer el agente, asociando cada funcionalidad con un comportamiento para permitir la selección del tipo de comportamiento, para que de esta manera JADE sea el encargado de la cronometrización.

Además cada agente tiene una cola de comportamientos activos, cada vez que se finalice uno, el cronometrizador de acuerdo con el tipo de comportamiento, decide si lo saca de la cola o lo ubica al final. De esta manera cuando se recibe un nuevo mensaje, éste se saca de esa cola y se ubica al final de la cola de comportamientos activos.

JADE posee un entorno gráfico que permite realizar algunas tareas de inspección y ejecución de agentes.

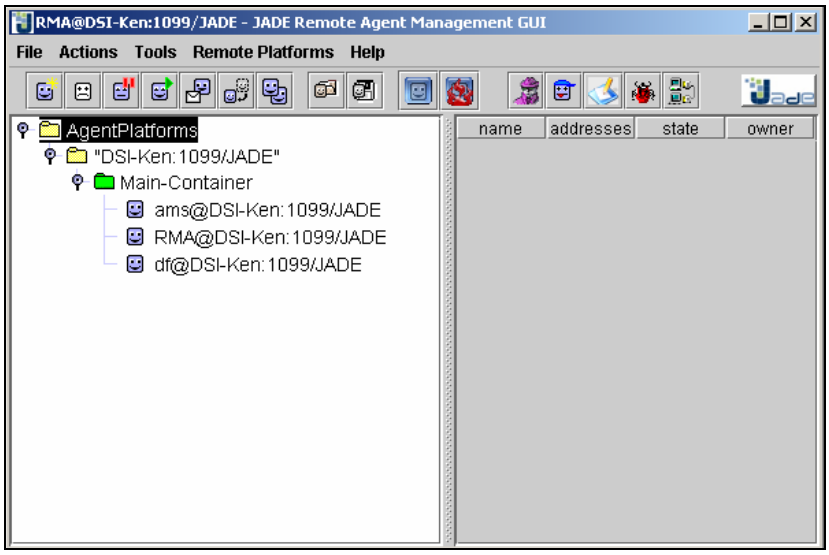

Figura 10. Entorno gráfico de JADE

6.4 ONTOLOGIAS, AGENTES Y LENGUAJES DE COMUNICACION

El término ontología hace referencia a la formulación de un esquema conceptual dentro de un dominio[34] dado, con la finalidad de facilitar la comunicación y la compartición de

[34] En términos de ontología se refiere a un área específica de interés

información entre diferentes sistemas. En otros términos, son vocabularios comunes para las personas y aplicaciones que trabajan en un entorno específico.

Desde un punto de vista más técnico las ontologías son colecciones de enunciados redactados en un lenguaje que define las relaciones entre conceptos y especifica reglas lógicas para razonar con ellos.

Las ontologías proceden del campo de la Inteligencia Artificial y de acuerdo a ésta una ontología es una descripción formal de los conceptos que se encuentran en un dominio de interés (clases o conceptos), en donde en cada una de éstos se describen las características y atributos de cada concepto (roles o propiedades) y sus restricciones (facetas[35] o restricción de roles).

En términos prácticos, desarrollar una ontología requiere:

- ✓ Definir las clases de la ontología.
- ✓ Organización de las clases con una taxonomía jerárquica (subclases - superclases).
- ✓ Definición de los roles o propiedades describiendo los valores permitidos por éstos.
- ✓ Definir las instancias para cada uno de los roles.

Algunas de sus características son:

- ✓ *Multiplicidad de ontologías*: Posibilidad de combinar dos o más de ontologías, en donde cada una de ellas puede introducir conceptualizaciones específicas.

- ✓ *Niveles de abstracción*: Capacidad de proporcionar una topología, en donde sea posible caracterizar una red de ontologías usando multiplicidad y abstracción.

- ✓ *Multiplicidad de representación*: Las ontologías pueden representar un concepto de muchas formas, por lo que pueden coexistir múltiples representaciones de éste.

[35] Se utiliza para definir qué tipo de valor puede contener un rol particular

- ✓ *Mapeo*: Permite crear relaciones entre los elementos de una o más ontologías para establecer conexiones, especializaciones, generalizaciones, etc.

- ✓ *Reutilización:* Permitir la reutilización del conocimiento perteneciente a un dominio a la hora de iniciar la elaboración de una ontología.

El uso de las ontologías se extiende desde la representación del conocimiento hasta una gran variedad de software, en donde la ontología es utilizada en múltiples propósitos, incluyendo el razonamiento inductivo[36], la clasificación, y una variedad de técnicas de resolución de problemas. En Internet las ontologías pueden representar:

- ✓ Productos, catálogos, recursos humanos
- ✓ Material educativo
- ✓ Servicios
- ✓ Plataforma, dispositivos
- ✓ Perfil del usuario

Algunas de las áreas de aplicación en las que las ontologías están involucradas son:

- ✓ Ingeniería del conocimiento
- ✓ Gestión del conocimiento
- ✓ Procesamiento del lenguaje natural
- ✓ Sistemas de información cooperativos
- ✓ Integración inteligente de información
- ✓ Recuperación de información
- ✓ Comercio electrónico

[36] Consiste en obtener conclusiones generales a partir de premisas que contienen datos particulares

6.4.1 CLASIFICACION

- ✓ **De acuerdo al grado de formalidad**

Se encuentran las *informales*, que se expresan directamente en cualquier lenguaje natural y pueden contener términos ambiguos o inconsistentes. Las *semi-informales* se expresan en una forma estructurada y restringida de algún lenguaje natural y permiten que las aplicaciones puedan usar los conceptos del dominio y sus relaciones pero pueden contener términos ambiguos o inconsistentes.

Y las *formales,* expresadas en lenguajes estructurados y definen los términos mediante lenguajes lógico - matemáticos cuyos símbolos se definen exactamente y sin ambigüedades. En consecuencia, estas ontologías permiten emplear teoremas y demostraciones. Las aplicaciones pueden usar los conceptos del dominio y sus relaciones pero no admiten términos ambiguos o inconsistentes.

- ✓ **De acuerdo con su dependencia y relación con una tarea específica desde un punto de vista**

Se encuentran las de *alto nivel o genéricas*, que describen conceptos generales como el espacio, tiempo, materia, objeto, etc. De *dominio*, que describen un vocabulario relacionado con un dominio genérico. De tareas o de técnicas básicas, que describen una tarea, actividad o artefacto. Y de *aplicación*, las cuales describen conceptos que dependen tanto de un dominio específico como de una tarea específica y generalmente son una especialización de ambas. Representan las necesidades de los usuarios relacionados con una aplicación específica.

- ✓ **Otras clasificaciones**

Entre otras clasificaciones encontramos las *terminológicas*, que detallan los términos que son usados para representar el conocimiento en el dominio. Suelen ser usadas para unificar vocabulario en un campo determinado. De *información,* que especifican la estructura de almacenamiento de bases de datos y ofrecen un marco para el almacenamiento estandarizado de información.

De *modelado de conocimiento*, la cual define conceptualizaciones del conocimiento. Contienen una rica estructura interna y suelen estar ajustadas al uso particular del conocimiento que describen.

6.4.2 LENGUAJES DE DEFINICION DE ONTOLOGIAS

Es importante resaltar que se puede utilizar cualquier lenguaje de programación para definir las ontologías.

El XML[37] proporciona una forma de escribir datos que es independiente de lenguajes, plataformas y herramientas, y que proporciona una estructura sintáctica para que los datos puedan ser interpretados por computadoras. De igual forma, XMLS, lenguaje de esquemas de XML, permite la definición de gramáticas y etiquetas significativas para los documentos a través de *namespaces* o espacios de nombre[38]. Sin embargo, XML o XMLS no son suficientes ya que su aporte radica en la estructura y no en la semántica[39]. Por esta razón es necesario utilizar más expresividad en el procesamiento semántico y en esta búsqueda surgieron los lenguajes especializados para tal fin, entre los que se encuentran principalmente RDF, OWL y DAML.

- ✓ **RDF – Resource Description Framework [29]**

Este modelo se basa en la idea de convertir las declaraciones de los recursos en expresiones con la forma sujeto – predicado – objeto y que en términos de RDF son conocidas como tripletes. El sujeto es el recurso, es decir aquello que se está describiendo. El predicado es la propiedad o relación que se desea establecer acerca del recurso. Por último, el objeto es el valor de la propiedad o el otro recurso con el que se establece la relación.

[37] Lenguaje de Etiquetado Extensible
[38] Los namespaces son un medio para organizar clases dentro de un entorno, agrupándolas de un modo más lógico y jerárquico.
[39] La semántica estudia cómo los símbolos se refieren a los objetos.

RDF está diseñado especialmente para representar metadatos[40] sobre recursos Web, tales como el título, autor, modificaciones de los datos de la página Web, copyright y otras licencias de información sobre documentos Web, así como la disponibilidad para algunos recursos compartidos. Es un modelo de datos para objetos o recursos.

También puede usarse para representar información sobre ciertas cosas que pueden ser identificadas en la Web, aunque no puedan ser directamente recuperadas en la misma, por ejemplo, información sobre artículos disponibles desde servicios online como información sobre especificaciones, precios, disponibilidad, etc. o la descripción de las preferencias de los usuarios de la Web para obtener información.

La combinación de RDF con otras herramientas como RDF Schema y OWL permite añadir significado a las páginas, y es una de las tecnologías esenciales de la Web semántica[41]. También se encuentra el modelo de metadatos Dublin Core o el formato RSS (RDF Site Summary) que permite distribuir titulares de noticias y contenidos por Internet de forma automatizada.

- ✓ **OWL – Web Ontology Language [28]**

OWL está diseñado para usarse cuando la información contenida en los documentos necesita ser procesada por programas o aplicaciones. Su desarrollo se ha visto impulsado con la puesta en marcha de la Web Semántica y por ende puede utilizarse para representar explícitamente el significado de términos en vocabularios y las relaciones entre aquellos términos.

OWL es una extensión del lenguaje RDF y emplea las tripletas de éste, pero con más poder expresivo, ya que posee más funcionalidades para expresar el significado y

[40] Un metadato es un dato estructurado sobre la información, o sea, información sobre información, o de forma más simple, datos sobre datos. Los metadatos en el contexto de la Web, son datos que se pueden guardar, intercambiar y procesar por medio del computador y que están estructurados de tal forma que permiten ayudar a la identificación, descripción clasificación y localización del contenido de un documento o recurso Web y que, por tanto, también sirven para su recuperación.

[41] Es la idea de añadir metadatos semánticos a la World Wide Web. Estas informaciones adicionales que describen el contenido, el significado y la relación de los datos, deben ser dadas en forma formal, así que es posible evaluarlas automáticamente por máquinas. La finalidad es mejorar la World Wide Web para ampliar la interoperabilidad entre los sistemas informáticos y reducir la mediación de operadores humanos.

semántica ofreciendo la posibilidad de representar contenido de la Web interpretable por máquina.

Este lenguaje tiene 3 sub-lenguajes que incrementan su expresión: OWL Lite, OWL DL, y OWL Full.

- ✓ **KIF – Knowledge Interchage Format [31]**

Lenguaje creado para representar ontologías basadas en la lógica de primer orden y permitir el intercambio de conocimiento entre diferentes sistemas, creados por distintos programadores, en distinta época y diferentes lenguajes.

No está diseñado para interactuar con humanos, pero puede utilizarse para este propósito en la forma más apropiada que encuentren las aplicaciones como por ejemplo Prolog, lenguaje natural, etc.

KIF no está previsto para la representación interna de conocimiento entre computadores o en un grupo de éstos. Di se leen datos basados en KIF, éstos son convertidos a la estructura interna programada (punteros, estructuras, arreglos, etc) y la computación que se realice se hace mediante estas estructuras. Cuando el sistema necesita comunicarse con otro se realiza el proceso inverso, convierte la información a datos de tipo KIF.

Las características esenciales de KIF son:

- ✓ El lenguaje tiene semántica declarativa. Es posible comprender el significado de las expresiones en si utilizar un interprete. De esta forma KIF difiere de otros lenguajes que están basados en intérpretes específicos como el Emycin y Prolog.

- ✓ El lenguaje posee comprensión lógica

- ✓ Provee una representación de conocimiento basado en conocimiento. Esto permite al usuario generar conocimiento explícito e introducir nuevo sin cambiar el lenguaje.

- ✓ **KQML – Knowledge Query Manipulation Language**

"Aunque es posible diseñar un marco de trabajo completo para la comunicación en el que todos los mensajes tengan la forma de oraciones en KIF, esto sería ineficiente, ya que se requeriría incluir información implícita sobre el agente que envía el mensaje y sobre el que lo recibe, debido a que la semántica de KIF es independiente del contexto. La comunicación se hace más eficiente si se provee una capa lingüística en la que el contexto se toma en cuenta".[42]

KQML fue concebido como un formato de mensajes y como un protocolo que maneja los mensajes para permitir a un programa identificar, conectarse e intercambiar información con otros programas. En términos lingüísticos se puede decir que KQML se enfoca principalmente a la parte pragmática de la comunicación.

Es independiente de la sintaxis del contexto, los protocolos de transporte y de alto nivel y asume un modelo de agentes que implica poseer entidades de alto nivel con capacidades cognitivas (base de conocimiento) y una descripción de nivel intencional.

Los mensajes enviados en KQML informan la naturaleza del contenido que transportan: pregunta, solicitud, confirmación, lo cual hace que cada mensaje represente un "acto de habla".

Internamente utiliza tres niveles:

- ✓ *Nivel de contenidos:* Lleva el contenido del mensaje en el lenguaje de representación propio de los agentes y por ende no es procesado por la implementaciones de KQML.

[42] Genesereth, 1992

- ✓ *Nivel de mensajes:* Utiliza el núcleo del lenguaje KQML, el cual determina los tipos de interacción que puedan tenerse con un agente que utilice KQML. Además identifica el protocolo e indica el lenguaje de contenidos y las ontologías asumidas.

- ✓ *Nivel de comunicaciones:* Trata los parámetros de comunicación de más bajo nivel como la identidad del emisor y receptor.

Los mensajes KQML se representan mediante performativas que constan de una lista de parejas del tipo atributo – valor.

6.4.3 FIPA ACL COMO LENGUAJE DE COMUNICACIÓN DE AGENTES

ACL es un lenguaje que esta formado por tres componentes: un vocabulario, un lenguaje de contenido llamado KIF y un lenguaje de comunicación llamado KQML.

Un mensaje de ACL es un mensaje KQML que consiste de una directiva de comunicación y un contenido semántico en KIF expresado en términos del vocabulario.

FIPA (Foundation for Intelligent Physical Agent) es una organización internacional sin ánimo de lucro dedicada a establecer un marco común y genérico para la construcción de agentes inteligentes y sistemas multi – agente, con el fin de conseguir interoperabilidad entre sistemas de agentes construidos por diferentes equipos.

Uno de los documentos más importantes de esta organización es el que define la Arquitectura Abstracta FIPA, cuyo objetivo, tal cual como aparece en el documento, es “permitir la construcción de sistemas que se integren con su entorno de computación particular, mientras que interoperan con sistemas de agentes que residen en entornos heterogéneos, todo con el mínimo esfuerzo”.

En FIPA la comunicación entre agentes se realiza mediante el intercambio de mensajes, los cuales representan “actos de habla” y están codificados en un lenguaje de comunicación de agentes.

También ofrece servicios de soporte como directorio y transporte de mensajes, los cuales se pueden implementar mediante agentes o software que se accede mediante métodos JAVA, C++, etc.

Arquitectura abstracta de FIPA [33]: Está compuesta por tres componentes: mensajería, directorio y ACL.

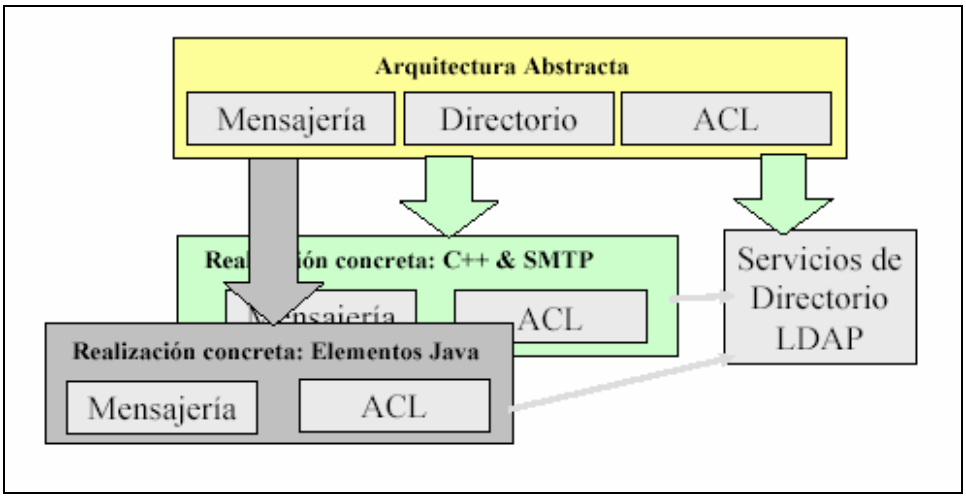

Figura 11. Arquitectura abstracta de FIPA

El servicio de directorio está formado por varias tuplas, cada una formada por dos parejas clave – valor:

- ✓ *Nombre del agente:* Identificador único global.
- ✓ *Localizador:* Dirección de transporte específica para comunicarse con el agente.

El servicio de mensajería también se compone de tuplas clave – valor, escritas en un lenguaje de comunicación de agentes (por ejemplo FIPA ACL) cuyo contenido está

expresado en lenguajes de contenidos (KIF), el cual puede hacer referencia a una ontología.

Los mensajes pueden contener embebidos otros mensajes y se incluyen el nombre del emisor y receptor.

```
( inform
            :sender agente1
            :receiver agente2
            :content
              (precio libro 1000)
            :in-reply-to round-4
            :reply-with bid04
            :language sl
            :ontology ontolibro
)
```

Figura 12. Ejemplo de un mensaje ACL

FIPA ACL está basado en “actos de habla” cuya semántica se basa en aptitudes mentales como creencias, intensiones, etc. Los mensajes son acciones comunicativas con una sintaxis similar a KQML.

Las conversaciones entre agentes siguen determinados patrones o secuencias típicas de mensajes, lo que se denomina protocolo de conversación. Los protocolos básicos de FIPA son:

FIPA – request
FIPA – query
FIPA – request-when
FIPA – contract-net
FIPA – iteraterated-contract-net
FIPA – auction-english

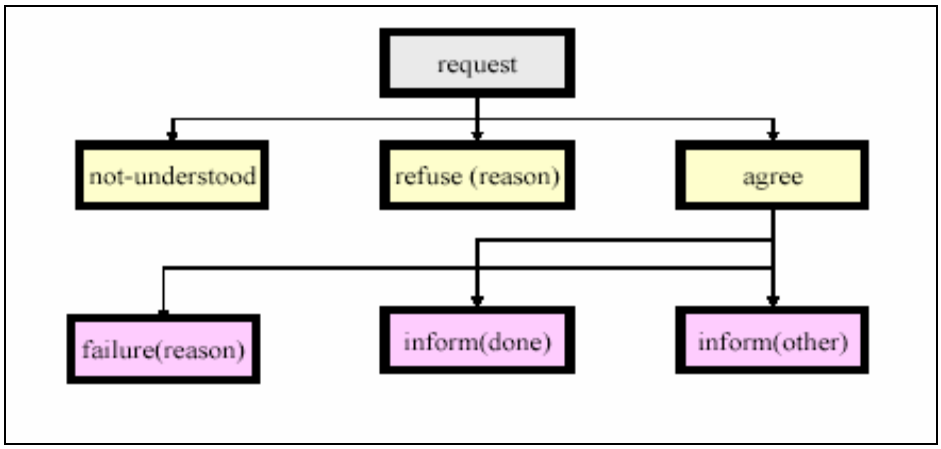

Figura 13. Protocolo FIPA – request

La semántica del lenguaje (SL) es una lógica multimodal con operadores modales para creencias, deseos, creencias falsas e intenciones u objetivos persistentes. Con ésta se puede representar proposiciones, objetos y acciones.

La semántica de cada acto comunicativo de FIPA ACL se define como un conjunto de fórmulas SL que describen:

- ✓ *Las precondiciones de factibilidad (FP):* Condiciones necesarias para el emisor, pero sin garantizar el acto comunicativo, ya que el agente no está obligado a realizarlo, inclusive si se cumplen las condiciones. Es decir, el agente tiene la facultad de elegir si lo realiza o no.

- ✓ *El efecto racional (RE):* Eventos esperados por el agente como resultado de realizar una acción. El agente receptor no garantiza el efecto esperado por el agente emisor, ya sea por las decisiones que tome o por que no pueda realizar la acción solicitada.

6.5 METODOLOGÍA DE DISEÑO DE AGENTES

Recientemente se menciona en muchos documentos acerca de lo qué se puede entender por un agente y las características que debe tener: autonomía, reactividad, iniciativa, habilidad social, etc. Tomando como base toda esta conceptualización surge ***INGENIAS,*** asumiendo la existencia de diferentes niveles de abstracción en el diseño de sistemas, entre ellos el nivel de símbolos y el nivel de conocimiento.

Al igual que un programa tradicional maneja símbolos, en el nivel de símbolos, un agente, que equivale a un programa en el nivel del conocimiento, maneja únicamente conocimiento y se comporta de acuerdo con el principio de racionalidad. Este principio estipula que un agente emprende acciones porque está buscando satisfacer un objetivo [36].

INGENIAS se centra en el desarrollo de Sistemas Multiagente (SMA) partiendo de modelos cercanos a las prácticas de ingeniería. Esta metodología propone un lenguaje de especificación de SMA así como su integración en el ciclo de vida. Dicha integración se consigue definiendo un conjunto de entregas y actividades involucradas en el desarrollo. Además plantea la división del problema en aspectos más concretos para generar diferentes vistas del sistema. La definición de las vistas se basa en la especificación de meta-modelos.

En cuanto a los casos de uso, en *INGENIAS,* expresan las interacciones que se pueden producir entre agentes y las tareas que se desarrollan en el sistema.

Esta metodología incorpora herramientas de soporte, es independiente de las plataformas de desarrollo de agentes y permite interactuar con plataformas de desarrollo de libre distribución.

A continuación se muestra la estructura de la metodología *INGENIAS* propuesta en [38]:

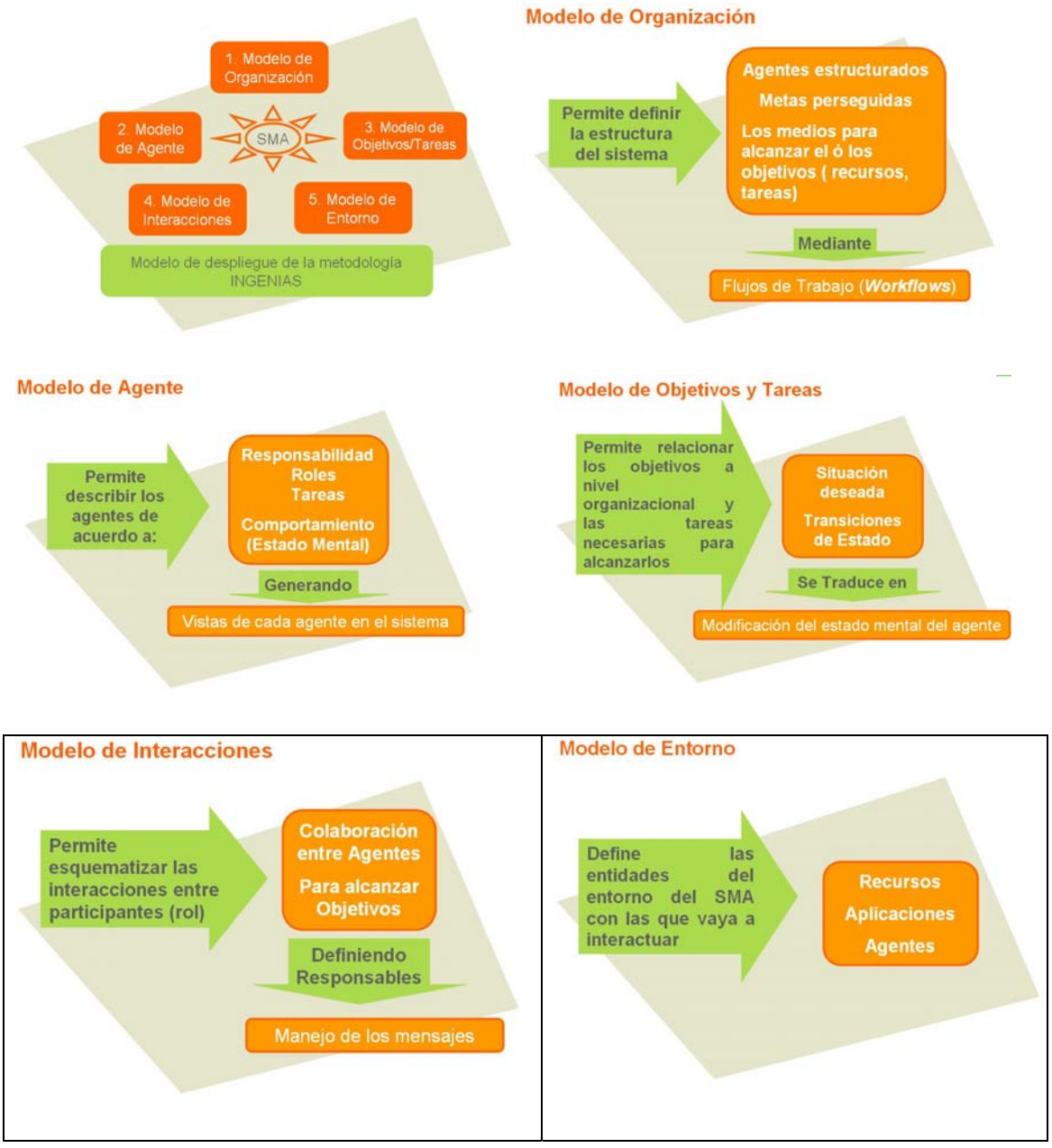

Figura 14. Estructura de la metodología INGENIAS

7. DISEÑO E IMPLEMENTACIÓN DE UN AGENTE PARA LA ADMINISTRACIÓN DE SERVIDORES WEB BASADOS EN TOMCAT

El agente a desarrollar tiene como objetivo implementar un servicio de administración del servidor Tomcat, el cual consiste en mantener activo y funcionando el sitio Web de la UIS.

En la siguiente figura se presenta la arquitectura del sistema multiagente sobre el que funcionará el agente propuesto.

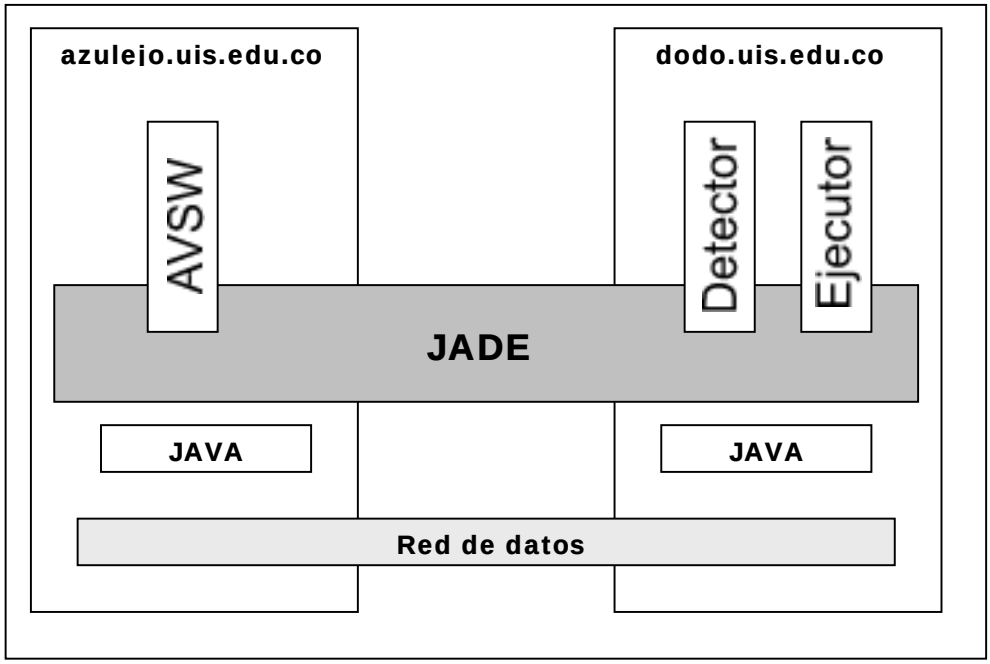

Figura 15. Arquitectura del sistema multiagente propuesto

7.1 ANALISIS DE REQUISITOS

Para plantear los requisitos se hace necesario familiarizarse con la estructura y entorno de trabajo del medio en el que el agente ejercerá su labor. En este caso la División de Servicios de Información de la Universidad Industrial de Santander.

El sitio Web de la Universidad se encuentra alojado en una máquina denominada dodo.uis.edu.co con las siguientes características:

DELL PowerEdge 1600SC
4 GB de memoria RAM
Disco duro de 60GB
Sistema operativo Linux Red Hat 9.0
Apache Tomcat (Apache 2.0.52, Tomcat 5.0.28)
SSL de 128 bits certificado

Además se cuenta con otro servidor (azulejo.uis.edu.co) que le sirve de respaldo a dodo.uis.edu.co en las épocas de más concurrencia de la página, alojando ciertas aplicaciones para balancear las cargas y agilizar las consultas.

En cuanto a la ejecución del sitio Web de la Universidad, se cuenta con dos servidores Web conocidos: Apache y Tomcat, trabajando los dos en conjunto y dividiéndose la carga, de tal manera que el servidor Apache atienda las solicitudes de las páginas estáticas y Tomcat de las páginas dinámicas. Lo anterior obliga a ejecutar los dos servidores al mismo tiempo para su correcto funcionamiento.

Aunque Apache se hace cargo de la parte estática de la Web de la Universidad, Tomcat es el servidor que mas uso hace de la memoria y de la concurrencia, debido a que la Web institucional cuenta con muchas aplicaciones dinámicas que acarrean un gran número de usuarios concurrentes y por ende Tomcat es el que puede, en ciertas ocasiones, verse colapsado por todas estas peticiones, generando un mal funcionamiento del sistema y en el peor de los casos su caída total.

En los casos en que el sitio Web empieza a colapsar, es donde el administrador debe ingresar al sistema y verificar el estado del servidor, que para el caso de la Universidad se identifican claramente dos situaciones que indican mal funcionamiento del Servidor:

- ✓ *El servidor está caído:* Se comprueba ejecutando el comando Unix **ps –fe|grep java** para comprobar si el proceso está activo. Si no se obtiene ningún resultado se deduce la caída total del servidor. Para restablecer el sistema se debe ejecutar de nuevo la aplicación que activa el servidor.

- ✓ *El servidor se está ejecutando pero no responde:* Se comprueba ejecutando el comando **ps –fe|grep java** y si el resultado es el nombre del proceso Yakarta-tomcat-5.0.28, que indica que el servidor no está caído. En este caso se procede a analizar el historial de funcionamiento del proceso para identificar porque si el Servidor está arriba, éste no funciona. Para restablecer el sistema se debe matar el proceso vigente con el comando **kill** y volver a ejecutar la aplicación que activa el servidor.

Por lo anterior es indispensable que el agente realice las siguientes tareas:

- ✓ Verificar si el servidor está respondiendo.

- ✓ Si no se está ejecutando el servidor se debe verificar si el problema es una caída total. Si este es el caso se continúa con el paso 4.

- ✓ Si no esta caído entonces proceder a matar el proceso con el comando respectivo.

- ✓ Ejecutar la aplicación que sube el servidor.

Además de estas consideraciones se debe tener en cuenta factores ajenos a la ejecución del servidor, como la caída de la red, la simple desactivación del equipo u otros[43].

Como consecuencia, el agente además de realizar tareas de ejecución de programas, también debe estar soportado con una estructura capaz de diferenciar las anteriores situaciones y garantizar independencia de la máquina que ejecuta el Servidor.

En conclusión, los requisitos de diseño del agente se pueden resumir en las siguientes acciones:

- ✓ Verificación
- ✓ Detección
- ✓ Ejecución
- ✓ Eliminación de proceso

Teniendo presente estos requisitos se plantea la creación de un agente localizado en azulejo.uis.edu.co encargado de tomar las decisiones y dos agentes en dodo.uis.edu.co encargados de detectar y activar el Servidor.

El agente ubicado en azulejo.uis.edu.co se identificará con el nombre de **AVSW** (Agente Verificador del Servidor Web) y los agentes en dodo.uis.edu.co se denominarán **detector** y **ejecutor.** Estos nombres se deben a la naturaleza de la acción que realizan; los agentes detector y ejecutor realizan acciones que el agente avsw les ordena de acuerdo a la información que intercambian.

De esta manera se establece inicialmente una jerarquía en cuanto a la importancia, siendo el agente ***AVSW*** el que toma las decisiones y por ende el agente central del sistema. Los otros dos se convierten en agentes dependientes de AVSW sin capacidad de decisión.

[43] Como corte de corriente eléctrica, daño en equipos de comunicación de la red, etc. Estos factores tienen una probabilidad muy baja de ocurrencia.

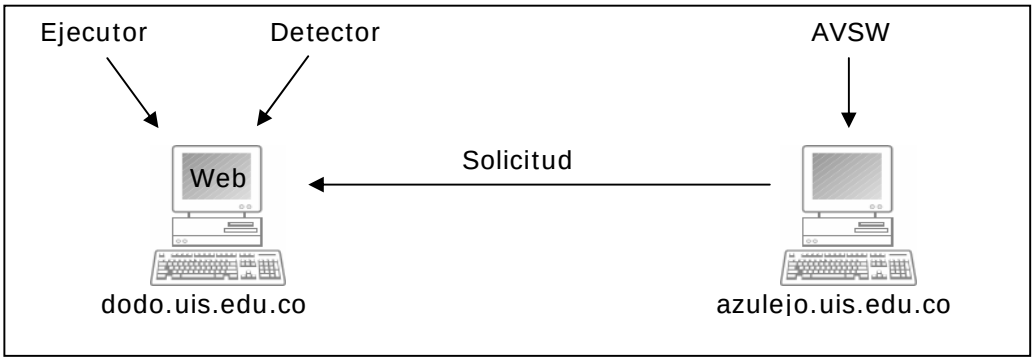

Figura 16. Esquema general del sistema multiagente

De acuerdo con lo planteado y haciendo uso de la metodología de diseño *INGENIAS* los casos de uso importantes para el sistema se resumen en la siguiente tabla:

Tabla 2. Casos de uso del sistema multiagente

Tarea	Descripción
Verificar si el Servidor está respondiendo	El agente AVSW envía un mensaje al Servidor para verificar si está activo.
Verificar la causa de la caída del Servidor	El agente Detector inspecciona e identifica si el proceso se encuentra inactivo o no responde.
Matar el proceso	Si el agente Detector identifica que el proceso se está ejecutando, procede a eliminar el proceso para garantizar una activación correcta.
Activar el Servidor	De acuerdo al mensaje enviado por el agente Detector, el agente AVSW le envía la orden al agente Ejecutor de activación del Servidor.

7.2 DEFINICION DE LAS REGLAS

Las reglas globales del sistema de agentes son:

Regla 1: El agente AVSW siempre debe estar solicitando al agente detector el estado del servidor.

Regla 2: El agente detector debe consultar el estado del servidor y retornar una respuesta de verdadero si el servidor está arriba y falso si el servidor está caído.

Regla 3: Si el estado del servidor es caído, el agente detector debe verificar si es una caída total o si es un problema de memoria en el servidor. Si es problema de memoria debe encontrar el número del proceso y eliminarlo, para que de esta forma se devuelva falso en cualquiera de los dos casos mencionados por esta regla.

Regla 4: Si la respuesta enviada por detector es falsa, AVSW debe solicitarle al agente ejecutor que suba nuevamente el servidor.

Regla 5: El agente ejecutor debe subir el servidor siempre y cuando AVSW se lo ordene y retornar verdadero si se realizó correctamente esta acción o falso de lo contrario.

Regla 6: Si el agente ejecutor retornó falso, AVSW debe solicitar al agente ejecutor que lo suba nuevamente, siempre y cuando no se supere el máximo número de intentos fallidos.

Regla 7: Todos los agentes deben estar ejecutándose en todo momento para garantizar la comunicación permanente entre ellos.

7.3 IMPLEMENTACION DEL AGENTE

Para implementar el sistema de agentes planteado, se debe crear como mínimo, un archivo JAVA por cada tipo de agente que vaya a estar presente en nuestro sistema, creando una clase derivada de la clase Agent de JADE. Además para definir la ontología se deben crear otros archivos JAVA adicionales.

7.3.1 Ontología

Una ontología en JADE es una instancia de la clase *jade.content.onto.Ontology* en la cual se definen los esquemas de la estructura de los predicados, acciones de los agentes y conceptos relevantes al dominio del problema. Estos esquemas son instancias de las clases *PredicateSchema*, *AgentActionSchema* y *ConceptSchema* incluidas en el paquete *jade.content.schema*. Además, para cada uno de los elementos que se definan en la ontología se deberá crear una clase asociada.

Para cada uno de los esquemas se definen una serie de campos, en donde cada uno de éstos corresponde con una variable de la clase correspondiente al esquema, y por lo tanto tendrán asociado un tipo.

De acuerdo con lo anterior, los elementos necesarios para definir la ontología son:

- ✓ Un concepto llamado servidor que contiene los siguientes campos:
  - IP: Dirección Ip del servidor que envía la solicitud.
  - Nombre: Nombre del servidor que envía la solicitud.
  - Acción: Acción que el servidor solicita a otro agente.
  - Respuesta: Respuesta de la acción realizada por el agente.

- ✓ Un predicado llamado conexión que hace referencia al concepto servidor.

Teniendo claros los elementos que van a formar parte de la ontología, se procede a crear el archivo java que contendrá el objeto de la clase *Ontology*, el cual será el encargado de contener la estructura de dichos elementos. En este caso el archivo queda codificado de la siguiente manera:

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

public class ontologia extends Ontology
{
  public static final String ONTOLOGY_NAME = "ontologia de conexion";
```

```
// Vocabulario
public static final String SERVIDOR_ = "Servidor";
public static final String SERVIDOR_IP = "ip";
public static final String SERVIDOR_NOMBRE = "nombre";
public static final String SERVIDOR_ACCION = "accion";
public static final String SERVIDOR_RESPUESTA = "respuesta";

public static final String CONEXION = "Conexion";
public static final String CONEXION_SERVIDOR = "Servidor";

// La instancia de esta ontología
private static Ontology instancia = new ontologia();

// Método para acceder a la instancia de la ontología
public static Ontology getInstance()
{
  return instancia;
}

// Constructor privado
private ontologia()
{
  // la ontologia extiende la ontología básica
  super(ONTOLOGY_NAME, BasicOntology.getInstance());

  try
  {
  add(new ConceptSchema(SERVIDOR_), servidor.class);
  add(new PredicateSchema(CONEXION), conexion.class);

  // Estructura del esquema para el concepto SERVIDOR_
  ConceptSchema cs = (ConceptSchema) getSchema(SERVIDOR_);
  cs.add(SERVIDOR_IP,(PrimitiveSchema) getSchema(BasicOntology.STRING),
          ObjectSchema.OPTIONAL);

       cs.add(SERVIDOR_NOMBRE,(PrimitiveSchema)
              getSchema(BasicOntology.STRING),ObjectSchema.OPTIONAL);

  cs.add(SERVIDOR_ACCION,(PrimitiveSchema)
              getSchema(BasicOntology.STRING),ObjectSchema.OPTIONAL);

  cs.add(SERVIDOR_RESPUESTA,(PrimitiveSchema)
              getSchema(BasicOntology.STRING),ObjectSchema.OPTIONAL);

       // Estructura del esquema para el predicado CONEXION
  PredicateSchema ps = (PredicateSchema) getSchema(CONEXION);
  ps.add(CONEXION_SERVIDOR, (ConceptSchema) getSchema(SERVIDOR_));
```

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

Como se mencionó anteriormente cada esquema incluido en la ontología está asociado con una clase Java, en este caso servidor.class:

```
import jade.content.Concept;

public class servidor implements Concept
{
   private String ip;
   private String nombre;
   private String accion;
   private String respuesta;

   public String getrespuesta()
   {
      return respuesta;
   }

   public String getaccion()
   {
      return accion;
   }

   public String getip()
   {
      return ip;
   }

   public String getnombre()
   {
      return nombre;
   }

   public void setrespuesta(String r)
   {
      respuesta = r;
   }

   public void setaccion(String acc)
```

```
    {
       accion = acc;
    }

    public void setip(String i)
    {
       ip  = i;
    }

    public void setnombre(String n)
    {
       nombre  = n;
    }
}
```

7.3.2 Agentes

Cada agente del sistema será una instancia de una clase java que sea subclase de la clase *Agent*. Esta clase se compone básicamente de un método *setup*, que se ejecuta al iniciarse el agente y que contiene código de inicialización, incluyendo instrucciones que especificarán la ontología a utilizar y los comportamientos asociados del agente.

Además de este método setup se dispone de una clase interna a la clase del agente por cada uno de los comportamientos asociados al agente. Estos comportamientos básicamente se utilizan para el envío y recepción de mensajes y definición de los métodos de detección e inicialización del servidor Web.

El lenguaje de comunicación entre agentes lo rige la ontología diseñada, por lo cual un mensaje tendrá un contenido similar al siguiente:

(servidor ip:192.168.80.40 nombre:azulejo.uis.edu.co accion:deteccion respuesta:)

Este mensaje es transformado por JADE en una cadena que se incluye en el contenido del mensaje a transmitir. Esta conversión y operaciones de comprobación son llevadas

a cabo por un objeto content manager, que podemos encontrar en el interior de cada agente y acceder a él mediante el método *getContentManager()* de la clase *Agent*.

El *content manager* provee de métodos para acceder a la funcionalidad de conversión, pero esta funcionalidad suele ser delegada en una ontología y en un content language codec. Más específicamente la ontología valida la información a ser convertida desde un punto de vista semántico mientras que el codec realiza la traducción a cadenas siguiendo las reglas sintácticas del content language.

Como lo que se quiere es que los agentes utilicen la ontología diseñada, se debe hacer uso de los métodos *registerLanguage()* y *registerOntology* del *content manager* del agente.

Para entrar en detalle con los agentes a implementar, se implementa una clase llamada solicitud dentro del agente detector, que hereda características de la clase SimpleBehaviour, lo que indica que el agente es del tipo reflejo simple. Esta clase es la encargada de recibir mensajes y de acuerdo a la acción requerida ejecutar la acción asociada.

La clase solicitud implementa un método llamado servidor_arriba() que retorna verdadero si el servidor está arriba y falso en caso contrario.

Este agente siempre está recibiendo solicitudes del agente avsw; cuando éste recibe la acción **detectar**, se hace un llamado al método servidor_arriba() y de acuerdo a la respuesta recibida envía al agente avws, que está ubicado en la máquina que la variable nombre indique, la respuesta arriba o abajo.

En este agente solo se ejecuta un evento, ya que su única función es la detección del estado del servidor.

La siguiente es la codificación de este agente:

```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.DFService;
import jade.domain.FIPAException;

import jade.content.*;
import jade.content.lang.*;
import jade.content.lang.sl.*;
import jade.content.onto.*;
import java.io.*;
import java.net.*;

public class detector extends Agent
{
    static boolean matando_proc = false;
   private Codec codec = new SLCodec();
   private Ontology onto = ontologia.getInstance();

   // Clase que describe el comportamiento que permite recibir un mensaje y
   // contestarlo

   class solicitud extends SimpleBehaviour
   {
   private boolean finalizado = false;

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

      public void action()
      {
         MessageTemplate mt = MessageTemplate.and(
         MessageTemplate.MatchLanguage(codec.getName()),
         MessageTemplate.MatchOntology(onto.getName()));
         ACLMessage mensaje = receive(mt);

      try
      {
         if(mensaje != null)
         {
               if(mensaje.getPerformative() == ACLMessage.NOT_UNDERSTOOD)
               {
                  System.out.println("Mensaje NOT UNDERSTOOD recibido");
            }
```

```
else
{
if(mensaje.getPerformative()== ACLMessage.INFORM)
{
      ContentElement ce =
         getContentManager().extractContent(mensaje);

      if (ce instanceof conexion)
      {
      // Recibido un QUERY_REF con contenido correcto
      conexion con = (conexion) ce;
      servidor serv = con.getservidor();

        if (serv.getaccion().equals("detectar"))
        {
            //Nombre del agente al que debe enviarse la respuesta
            ACLMessage respuesta = new
               ACLMessage(ACLMessage.INFORM);

            respuesta.setSender(getAID());
            AID identificador =
               new
               AID((String)"avswo@"+serv.getnombre()+":1099/JADE");

            identificador.addAddresses
               ("http://"+serv.getnombre()+":7778/acc");

            respuesta.addReceiver(identificador);

            respuesta.setLanguage(codec.getName());
            respuesta.setOntology(onto.getName());

            boolean serv_arr = servidor_arriba();

            if (serv_arr == true)
         {
            serv.setrespuesta("arriba");
            matando_proc = false;
         }
         else serv.setrespuesta("abajo");

         getContentManager().fillContent(respuesta,con);
         send(respuesta);
      }
      }
      else
      {
         // Recibido un QUERY_REF con contenido incorrecto
```

```
                    ACLMessage respuesta = mensaje.createReply();
                    respuesta.setPerformative(ACLMessage.NOT_UNDERSTOOD);
                    respuesta.setContent("Contenido incorrecto");
                    send(respuesta);
                }
            }
            else
            {
                // Recibida una performativa incorrecta
                ACLMessage respuesta = mensaje.createReply();
                respuesta.setPerformative(ACLMessage.NOT_UNDERSTOOD);
                respuesta.setContent("Performativa incorrecta");
                send(respuesta);
              }
            }
      }
      else
      {
            //System.out.println("No message received");
      }
      }
      catch (jade.content.lang.Codec.CodecException ce)
      {
      System.out.println("Error de codec : "+ce);
      }
      catch (jade.content.onto.OntologyException oe)
      {
         System.out.println("Error de ontologia : "+oe);
      }
}

public boolean done()
{
   return finalizado;
}

public boolean servidor_arriba() //Función que verifica si el servidor está arriba
{
 boolean respuesta = false;

 try
 {
  HttpURLConnection url = (HttpURLConnection)new
     URL("http://halcon1.uis.edu.co/estudiantes").openConnection();

  if (url.getResponseCode() == 200 &&
     url.getHeaderField("Server").startsWith("Apache"))
  {
```

```
          respuesta = true;
        }
        else
        {
           try
           {
              if (matando_proc == false)
              {
                  matando_proc = true;

                  java.lang.Runtime comando = java.lang.Runtime.getRuntime();
                 java.lang.Process proc = comando.exec(" rm -f
                    /usr/local/jade/jade/file_proc");

                 proc = comando.exec("./proceso.sh");

                BufferedReader file_proc= new BufferedReader(new
                  FileReader("/usr/local/jade/jade/file_proc"));
                  String salida = file_proc.readLine();
                  file_proc.close();
                  String num_proc = salida.substring(0,salida.indexOf(" "));
                  System.err.println("Proceso de Tomcat en ejecución   : "+salida);
                  System.err.println("Matando proceso de Tomcat número : "
                    +num_proc);

                  System.err.println("");

                  proc = comando.exec("kill -9 "+num_proc);
              }
           }
           catch(Exception e){}
         }
       }
       catch (IOException ioe)
       {
         respuesta = false;
       }
       return respuesta;
     }
  } //Fin de la clase solicitud

protected void setup()
{
    DFAgentDescription dfd = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("Agente Detector");
    sd.setName(getName());
    sd.setOwnership("DCCIA");
```

```
        dfd.setName(getAID());
        dfd.addServices(sd);

        try
        {
           DFService.register(this,dfd);
        }
        catch (FIPAException e)
        {
           System.err.println(getLocalName()+" El registro del DF no pudo concretarse.
              Razón: "+e.getMessage());

           doDelete();
        }

        getContentManager().registerLanguage(codec);
        getContentManager().registerOntology(onto);

        solicitud evento_solicitud = new  solicitud(this);
        addBehaviour(evento_solicitud);
     }
}
```

Es importante resaltar en el agente detector que método servidor_arriba() detecta si el servidor realmente está caído, de lo contrario utiliza el comando kill para matar el proceso.

El agente ejecutor es idéntico en estructura al agente detector. La diferencia radica en que este agente implementa un método llamado iniciar_servicio(), encargado de subir el servidor. La codificación es la siguiente:

```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.DFService;
import jade.domain.FIPAException;

import jade.content.*;
import jade.content.lang.*;
import jade.content.lang.sl.*;
import jade.content.onto.*;
import java.io.*;
```

```
import java.net.*;

public class ejecutor extends Agent
{
    static boolean matando_proc = false;
   private Codec codec = new SLCodec();
   private Ontology onto = ontologia.getInstance();

   // Clase que describe el comportamiento que permite recibir un mensaje y
   // contestarlo

   class solicitud extends SimpleBehaviour
   {
   private boolean finalizado = false;

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

      public void action()
      {
         MessageTemplate mt = MessageTemplate.and(
         MessageTemplate.MatchLanguage(codec.getName()),
         MessageTemplate.MatchOntology(onto.getName()));
         ACLMessage mensaje = receive(mt);

      try
      {
         if(mensaje != null)
         {
               if(mensaje.getPerformative() == ACLMessage.NOT_UNDERSTOOD)
               {
                  System.out.println("Mensaje NOT UNDERSTOOD recibido");
            }
               else
               {
               if(mensaje.getPerformative()== ACLMessage.INFORM)
               {
                     ContentElement ce =
                        getContentManager().extractContent(mensaje);

                     if (ce instanceof conexion)
                     {
                        // Recibido un QUERY_REF con contenido correcto
                        conexion con = (conexion) ce;
                        servidor serv = con.getservidor();
```

```
                if (serv.getaccion().equals("subir"))
                {
                    //Nombre del agente al que debe enviarse la respuesta
                    ACLMessage respuesta = new
                       ACLMessage(ACLMessage.INFORM);

                    respuesta.setSender(getAID());
                    AID identificador = new
                       AID((String)"avswo@"+serv.getnombre()+":1099/JADE");

                    identificador.addAddresses
                       ("http://"+serv.getnombre()+":7778/acc");
                    respuesta.addReceiver(identificador);

                    respuesta.setLanguage(codec.getName());
                    respuesta.setOntology(onto.getName());

                    boolean estado = iniciar_servicio();

                    if (estado == true) serv.setrespuesta("inicializando");
                    else serv.setrespuesta("falla");

                    getContentManager().fillContent(respuesta,con);
                  send(respuesta);
                  }
               }
               else
               {
                  // Recibido un QUERY_REF con contenido incorrecto
                  ACLMessage respuesta = mensaje.createReply();
                  respuesta.setPerformative(ACLMessage.NOT_UNDERSTOOD);
                  respuesta.setContent("Contenido incorrecto");
                  send(respuesta);
               }
         }
         else
         {
               // Recibida una performativa incorrecta
               ACLMessage respuesta = mensaje.createReply();
               respuesta.setPerformative(ACLMessage.NOT_UNDERSTOOD);
               respuesta.setContent("Performativa incorrecta");
               send(respuesta);
            }
         }
}
else
{
         //System.out.println("No message received");
```

```
        }
        }
        catch (jade.content.lang.Codec.CodecException ce)
        {
        System.out.println("Error de codec : "+ce);
        }
        catch (jade.content.onto.OntologyException oe)
        {
            System.out.println("Error de ontologia : "+oe);
        }
    }

    public boolean done()
    {
        return finalizado;
    }

    public boolean iniciar_servicio() //Función que inicializa el Tomcat
    {
        boolean respuesta = false;
        try
        {
            java.lang.Runtime comando = java.lang.Runtime.getRuntime();
            String tomcat_home = System.getenv("CATALINA_HOME");

            String subir = tomcat_home+"/bin/./startup.sh";

            java.lang.Process proc = comando.exec(subir);
            respuesta = true;
        }
        catch(Exception e)
        {
            System.err.println("No se pudo subir el servidor debido a : "+e.toString());
        }
        return respuesta;
    }
 } //Fin de la clase solicitud

protected void setup()
{
    DFAgentDescription dfd = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("Agente Ejecutor");
    sd.setName(getName());
    sd.setOwnership("DCCIA");
    dfd.setName(getAID());
    dfd.addServices(sd);
```

```
        try
        {
           DFService.register(this,dfd);
        }
        catch (FIPAException e)
        {
           System.err.println(getLocalName()+" El registro del DF no pudo concretarse.
              Razón: "+e.getMessage());

           doDelete();
        }

        getContentManager().registerLanguage(codec);
        getContentManager().registerOntology(onto);

        solicitud evento_solicitud = new  solicitud(this);
        addBehaviour(evento_solicitud);
    }
}
```

El agente AVSW es un poco mas complejo, ya que debe enviar y recibir solicitudes y de acuerdo a las respuestas recibidas indicarle al agente correcto la acción a ejecutar. Para ello implementa dos clases: solicitud y orden. La primera encargada del envío y recepción de mensajes al agente detector y la segunda al agente ejecutor.

Este agente siempre está en comunicación con el agente detector, es decir, AVSW conoce en todo momento el estado del servidor, y cuando éste deja de funcionar inmediatamente le envía la orden al agente ejecutor de iniciar servicios. Una vez restablecidos deja de comunicarse con el ejecutor.

La codificación es la siguiente:

```
import java.io.StringReader;
import java.io.OutputStreamWriter;
import java.io.BufferedWriter;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.InterruptedIOException;
import java.io.IOException;
import java.io.PrintWriter;
```

```
import java.text.*;
import java.io.*;

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.DFService;
import jade.domain.FIPAException;

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

public class avsw extends Agent
{
   static boolean enviada_solicitud = false;

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

  class solicitud extends SimpleBehaviour
  {
      private boolean finalizado = false;

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

      public void action()
      {
         try
         {
            //Nombre del otro agente con el que se comunica
            ACLMessage mensaje = new ACLMessage(ACLMessage.INFORM);
            mensaje.setSender(getAID());
            AID identificador = new
               AID((String)"detectoro@halcon1.uis.edu.co:1099/JADE");

            identificador.addAddresses ("http://halcon1.uis.edu.co:7778/acc");
            mensaje.addReceiver(identificador);

            mensaje.setLanguage(codec.getName());
            mensaje.setOntology(onto.getName());
```

```
            servidor serv = new servidor();
            serv.setip("192.168.37.165");
            serv.setnombre("azulejo.uis.edu.co");
            serv.setaccion("detectar");

            conexion con = new conexion();
            con.setservidor(serv);

            getContentManager().fillContent(mensaje,con);

            send(mensaje);
         }
         catch (jade.content.lang.Codec.CodecException ce)
         {
         System.out.println("Error de codificación : "+ce);
         }
         catch (jade.content.onto.OntologyException oe)
         {
            System.out.println("Error de ontología : "+oe);
         }
      }

      public boolean done()
      {
         return finalizado;
      }
   }
   // Fin de la clase solicitud
//-----------------------------------------------------------------------------

  class orden extends SimpleBehaviour
  {
      private boolean finalizado = false;
      private String mensaje_ant = "";

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

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

      public void escribir_logs(String mensaje)
      {
         try
         {
```

```
        java.util.Date tiempo=new java.util.Date();
        SimpleDateFormat formato_fecha=new SimpleDateFormat("dd MMMM
            yyyy");

        SimpleDateFormat formato_hora=new SimpleDateFormat("h:mm a");
        BufferedReader in= new BufferedReader(new
            FileReader("/usr/local/path_logs_jade.txt"));

        FileWriter fw = new FileWriter(in.readLine(),true);
        in.close();
        fw.write(formato_fecha.format(tiempo)+" " +
            formato_hora.format(tiempo)+" >> "+mensaje+"<br>");

        fw.close();
    }
    catch(Exception e)
    {
        System.err.println("Error en el m&eacute;todo escribir_logs : "+
            e.toString());
    }
}

public void action()
{
    MessageTemplate mt = MessageTemplate.and(
    MessageTemplate.MatchLanguage(codec.getName()),
    MessageTemplate.MatchOntology(onto.getName()));
    ACLMessage mensaje = receive(mt);

    try
    {
        java.util.Date tiempo=new java.util.Date();
        SimpleDateFormat formato_fecha=new SimpleDateFormat("dd MMMM
            yyyy");

        SimpleDateFormat formato_hora=new SimpleDateFormat("h:mm a");

       if(mensaje != null)
       {
            if(mensaje.getPerformative() == ACLMessage.NOT_UNDERSTOOD)
            {
                System.out.println("Mensaje NOT UNDERSTOOD recibido");
           }
            else
            {
            boolean servidor_arriba = false;
            int contador_envios = 0;
```

```
if(mensaje.getPerformative() == ACLMessage.INFORM)
{
   ContentElement ce = getContentManager().extractContent(mensaje);
      if (ce instanceof conexion)
      {
      conexion con = (conexion) ce;
      servidor serv = con.getservidor();

         if (serv.getrespuesta() != null)
         {
            if (!mensaje_ant.equals(serv.getrespuesta()))
            {
            System.err.println(formato_fecha.format(tiempo)+" "+
            formato_hora.format(tiempo)+" - "+serv.getrespuesta());

            System.err.println("Estado : "+serv.getrespuesta());

            mensaje_ant = serv.getrespuesta();

            if (serv.getrespuesta().equals("arriba"))
            {
               servidor_arriba = true;
               enviada_solicitud = false;
               contador_envios = 0;
            }

            if ((serv.getrespuesta().equals("abajo")) &&
               (enviada_solicitud == false))
            {
               servidor_arriba = false;

               if (enviada_solicitud == false)
               {
                  enviada_solicitud = true;

                  ACLMessage accion = new
                     ACLMessage(ACLMessage.INFORM);
                     accion.setSender(getAID());
                     AID identificador_accion = new
                     AID((String)"ejecutoro@halcon1.uis.edu.co:1099/JA
                     DE");

                     identificador_accion.addAddresses
                     ("http://halcon1.uis.edu.co:7778/acc");
                     accion.addReceiver(identificador_accion);

                     accion.setLanguage(codec.getName());
                     accion.setOntology(onto.getName());
```

```
                                    serv.setaccion("subir");
                                    getContentManager().fillContent(accion,con);
                                    send(accion);
                                }
                            }

                            if (serv.getrespuesta().equals("falla"))
                            {
                                contador_envios++;
                                if (contador_envios < 3) enviada_solicitud = false;
                                }
                            }
                        }
                    }
                }
            }
            else
            {
                //System.out.println("Ningun mensaje recibido");
            }
          }
          catch(Exception e)
          {
             System.out.println("Error : "+e.toString());
          }
        }

        public boolean done()
        {
          //Como queremos que siempre se ejecute devolvemos false
          return finalizado;
        }
      } //Fin de la clase orden
//---------------------------------------------------------------------------

   protected void setup()
   {

      /** Registrarse en el DF */
      DFAgentDescription dfd = new DFAgentDescription();
      ServiceDescription sd = new ServiceDescription();
      sd.setType("Solicitar deteccion");
      sd.setName(getName());
      sd.setOwnership("DCCIA");
      dfd.setName(getAID());
      dfd.addServices(sd);
```

```
      try
      {
         DFService.register(this,dfd);
      }
      catch (FIPAException e)
      {
         System.err.println(getLocalName()+" El registro del DF no pudo concretarse.
            Razón: "+e.getMessage());

         doDelete();
      }

      getContentManager().registerLanguage(codec);
      getContentManager().registerOntology(onto);

      solicitud evento_solicitud = new solicitud(this);
      addBehaviour(evento_solicitud);
      orden evento_orden = new orden(this);
      addBehaviour(evento_orden);
   }
}
```

7.3.3 Puesta en marcha

Para soportar el sistema se debe tener instalado JAVA versión superior a la 1.4, la cual puede ser descargada de la página de Sun Microsystems. También se debe instalar JADE, cuya versión actual es la 3.4 y se puede descargar de http://jade.cselt.it/. Además se deben tener en cuenta las variables de entorno que indiquen el directorio donde está instalado JAVA y la localización de las siguientes librerías de JADE:

- ✓ jade.jar
- ✓ jadeTools.jar
- ✓ crimson.jar
- ✓ Base64.jar
- ✓ iiop.jar
- ✓ http.jar

En el caso del sistema multiagente desarrollado se trabajó con JAVA versión 1.5 y JADE versión 3.4 y para ponerlo en funcionamiento se deben tener las siguientes consideraciones:

- ✓ Como todos utilizan una ontología personalizada, la clase que la referencia debe estar en todos los servidores que contienen agentes. En este caso dodo.uis.edu.co y azulejo.uis.edu.co.

- ✓ Al igual que la clase de la ontología también las clases conexión y servidor deben estar en todos los servidores.

- ✓ El agente AVSW solo debe estar en el computador donde no se encuentre el Servidor. En este caso azulejo.uis.edu.co.

- ✓ Los agentes detector y ejecutor deben estar en el computador donde se ejecuta el Servidor Web. En este caso dodo.uis.edu.co.

- ✓ La plataforma JADE debe estar instalada en todos los computadores que ejecuten agentes: dodo.uis.edu.co y azulejo.uis.edu.co.

Para iniciar la ejecución de los agentes se utiliza el siguiente comando:

En azulejo.uis.edu.co: java jade.Boot avsw:avsw

En dodo.uis.edu.co: java jade.Boot detector:detector ejecutor:ejecutor

De aquí en adelante los agentes son los encargados de restablecer los servicios siempre y cuando la caída del servidor se deba a las causas que se tuvieron en cuenta para el diseño.

8. CONCLUSIONES Y RECOMENDACIONES

De acuerdo al enfoque dado a la presente monografía, es importante resaltar la tarea que realizan los servidores Web y la responsabilidad que tienen en un sinnúmero de aplicaciones que hoy en día rigen el destino de muchas empresas; desde la simple inspección del catálogo de elementos ofrecido por proveedores hasta la decisión de realizar inversiones millonarias en banca extranjera.

Todo esto conlleva a reconocer a los servidores como herramienta fundamental de trabajo e inversión en las organizaciones y por ende su elección no debe estar basada en conceptos frívolos y superficiales y si más bien en un estudio detallado de su estructura y características especiales que puedan adaptarse de manera eficiente al entorno de trabajo de la empresa.

De igual forma, se puede apreciar la importancia que tiene el avance científico y tecnológico en la optimización de procesos mediante el uso de agentes inteligentes, facilitando a la organización no solo un mejor uso de sus recursos si no también tomando las mejores decisiones en cuanto a procedimientos específicos se refiere. Todo esto para ofrecerle al usuario final disponibilidad, agilidad y eficiencia en los sistemas que funcionan a través de la Web.

Pero no solo el uso de la tecnología de agentes inteligentes garantiza optimización, se debe tener claro que las decisiones que éstos toman, están basadas en un análisis del funcionamiento de la organización y sobre todo del diseño de interacciones establecida entre los agentes. Por esto es fundamental resaltar la importancia de las ontologías como lenguaje de comunicación, generando una gran diferencia entre sistemas de agentes sencillos y los grandes sistemas multiagente.

En cuanto al diseño del agente implementado, se debe resaltar la importancia que tiene el conocimiento de los procesos ejecutados, el porqué de las fallas, sus causas y lo más importante: las soluciones.

También es importante resaltar la plataforma JADE como base para el desarrollo de agentes inteligentes, proporcionando una arquitectura bien estructurada y de libre distribución, constituyéndose como una herramienta en continuo crecimiento.

El sistema de agentes implementados en esta monografía es sólo la solución a un problema, lo cual generó la creación de un sistema de agentes sencillo. Se puede decir que es la semilla de un gran sistema multiagente del futuro en la que muchos servicios implementados por otros agentes interactúan para ofrecer servicios Web ideales.

Por tal razón la recomendación es hacer crecer este sistema no solo a nivel de servidor si no también de switches y routers creando un grupo especial de desarrollo de agentes inteligentes que continuamente estén documentado y desarrollando servicios de optimización.

BIBLIOGRAFIA

Servidores Web

[1] http://www.cibernetia.com/manuales/instalacion_servidor_web/1_conceptos_basicos.php

[2] http://observatorio.cnice.mec.es/modules.php?op=modload&name=News&file=article&sid=366

[3] http://www.creaformacion.com/images/libros/Dreamweaver_8_vol1.pdf

[4] http://www.empresaweb.net/primeros_pasos.php

[5] http://www.htmlpoint.com/apache/index.html

[6] http://www.consol.org.mx/2004/material/123/Proyecto_Cherokee_-_CONSOL_2004.pdf

[7] http://es.wikipedia.org/wiki/Servidor_HTTP_Apache

[8] http://es.wikipedia.org/wiki/Servidor_web

[9] http://www.faqs.org/rfcs/rfc2616.html

[10] CHOPRA, Vivek, BAKORE, Amit, GALBRAITH, Ben, LI, Sing, WIGGERS, Chanoch. Professional Apache Tomcat 5. Wrox. Indianápolis. ISBN 0-7645-5902-8

Agentes inteligentes

[11]

PEÑA DE CARRILLO, Clara Inés. Sistemas Multiagente para el Tratamiento de la Información en el Web: Agentes de Interfaz, Agentes de Información, Agentes de Aprendizaje, Agentes Intermediarios. I Congreso Internacional de Ingeniería de Sistemas. Noviembre 13 – 17 de 2000.

[12]

PEÑA DE CARRILLO, Clara Inés. Tecnología de Agentes Inteligentes. Seminario. Abril de 2005. División de Servicios de Información.

[13] http://ciberconta.unizar.es/LECCION/INTRODUC/482.HTM

[14] http://strix.ciens.ucv.ve/~iartific/Material/ND_Agentes_Inteligentes.pdf#search=

%22agentes%20inteligentes%22

[15] http://cruzrojaguayas.org/inteligencia/Introducci%F3n1.htm

[16] http://www.wikilearning.com/agentes_inteligentes-wkc-5095.htm

[17] http://personales.upv.es/ccarrasc/doc/2000-2001/
Estudio%20Comp.%204%20ag.%20-%20Eva%20Campoy%20et%20al/
diapossrp.ppt#260,6,AGENTES%20INTELIGENTES

[18] http://www.ati.es/novatica/2000/145/vjulia-145.pdf#search=
%22acl%20agent%22

[19] http://www.robot.uji.es/docencia/II28/teoria/transparencias-tema02.pdf

[20] http://es.wikipedia.org/wiki/Ontolog%C3%ADa_(Inform%C3%A1tica)

[21] http://www.wshoy.sidar.org/index.php?2005/12/09/
30-ontologias-que-son-y-para-que-sirven

[22] http://es.geocities.com/recupdeinformacion_ontologias/sobreontolgias.htm

[23] http://fcom.uh.cu/websem/onto-objetivos-y-aplicaciones.htm

[24] http://protege.stanford.edu/publications/ontology_development/
ontology101-noy-mcguinness.html

[25] http://protege.stanford.edu/publications/OntologiesAndTools/index.htm

[26] http://ccc.inaoep.mx/~emorales/Cursos/RdeC/node208.html

[27] http://www.hipertexto.info/documentos/ontologias.htm

[28] http://www.hipertexto.info/documentos/owl.htm

[29] http://www.hipertexto.info/documentos/rdf.htm

[30] http://html.rincondelvago.com/agentes-moviles.html

[31] http://logic.stanford.edu/kif/dpans.html

[32] http://ants.dif.um.es/~juanbot/page_files/uimp2002.pdf#search=
%22acl%20agent%22

[33] http://www.fdi.ucm.es/profesor/jpavon/doctorado/acl.pdf

[34] http://jade.tilab.com/papers/2003/monografia.pdf

[35] http://es.wikipedia.org/wiki/Inteligencia_%28desambiguaci%C3%B3n%29

[36] http://grasia.fdi.ucm.es/ingenias/Spain/estado/index.php

[37] http://grasia.fdi.ucm.es/ingenias/Spain/index.php
[38]

PEÑA DE CARRILLO, Clara Inés, CRUZ RODRIGUEZ, Rubén Darío, DELGADO MARTINEZ, Ricardo, CARRILLO CAICEDO, Gilberto. Sistemas Multiagente para el Análisis de Mercados de Energía Eléctrica. 2006.