UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

Instituto de Ciencias Básicas e Ingeniería

Centro de Investigación en Tecnologías de Información y Sistemas

TESIS

"Cooperación en equipos de agentes expertos"

que para obtener el grado de Maestría en Ciencias Computacionales, presenta

Ana Leticia Palacios Coyoli

Director:
M. en C. Ramón Soto de la Cruz

Co-Director:
Dr. Gustavo Núñez Esquer

Pachuca de Soto, Hgo., Marzo de 2004.

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO
INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA
CENTRO DE INVESTIGACIÓN EN TECNOLOGÍAS DE INFORMACIÓN Y SISTEMAS

CITIS

Oficio No. CITIS-0136/2004

L. en Comp. Ana Leticia Palacios Coyoli
Presente

Por este conducto le comunico que el Jurado asignado para la revisión de su trabajo de tesis titulado **"Cooperación en Equipos de Agentes Expertos",** que para obtener el grado de Maestro en Ciencias Computacionales fue presentado por usted, ha tenido a bien, en

A continuación se integran las firmas de conformidad de los integrantes del Jurado:

PRESIDENTE:	**Dr. Joel Suárez Cansino**
PRIMER VOCAL:	**Dr. Guillermo Sánchez Díaz**
SECRETARIO:	**Dr. Omar López Ortega**
PRIMER SUPLENTE:	**Dr. Julio Waissman Vilanova**
SEGUNDO SUPLENTE:	**M. en C. Félix Agustín Castro Espinoza**

ATENTAMENTE
"AMOR, ORDEN Y PROGRESO"
Pachuca de Soto, Hidalgo, 18 de marzo de 2004

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO C. I. T. I.

Dr. Roberto A. Hernández Gómez
Coordinador de la Maestría en Ciencias Computacionales

C- c. p. M. en D. Adolfo **Pontigo Loyola**.- Director de Control Escolar
C. c. P. M. en C. Raúl García Rubio.- Director del ICBI
C. c. **p. Minutario/ apl**

Carretera Pachuca-Tulancingo Km 4.5
Ciudad Universitaria C.P. 42084
Pachuca de Soto, Hidalgo

Tel. (01771)71-72000 ext. 6738,6735
Fax. (01771)71-72000ext. 673
email: citis@uaeh.reduaeh.mx

Agradecimientos

Esta tesis es el resultado de varios años de trabajo realizado en el Centro de Investigación en Tecnologías de Información y Sistemas (CITIS), que me proporcionó los recursos necesarios para llevarla acabo. Agradezco especialmente al grupo de profesores que en su momento me tomaron en cuenta para que me fuera otorgada una beca durante los dos años de estudios del programa de maestría, lo cual permitió lograr mi objetivo. Además, agradezco a las siguientes personas por todo el apoyo recibido:

A mis sinodales, por todos los comentarios y observaciones que me hicieron en el proceso de revisión, y que sin lugar a duda me proporcionaron las bases para mejorar este trabajo.

A mis asesores, por la confianza y paciencia que tuvieron durante estos años de esfuerzo. Al *M. en C. Ramón Soto de la Cruz* le doy las gracias por su apoyo constante en todos aquellos problemas que se me presentaron al realizar esta tesis, incluyendo los existenciales, y además por compartir todas esas experiencias y consejos que enriquecieron muchos aspectos en mí, en lo profesional y personal. Al *Dr. Gustavo Núñez Esquer* agradezco enormemente todas aquellas horas de trabajo, de las que aprendí que el sacrificar ciertas cosas en la vida con provecho, nos pueden llevar a cosechar un gran fruto.

A las instancias administrativas, que de alguna forma tuvieron que ver con la finalización de este trabajo, ya que sin su apoyo los trámites administrativos correspondientes no se hubieran llevado a cabo de la mejor manera.

A todos mis profesores de la maestría, por todo el conocimiento aportado y ayuda incondicional, todos ustedes contribuyeron en la formación profesional que he logrado hasta ahora.

A mis compañeros, con los que compartí momentos de estudio, desvelos, discusiones y también de diversión, gracias por brindarme su amistad, comprensión y consejos que llevaré siempre conmigo.

Un invaluable reconocimiento y una dedicatoria muy especial de esta tesis *a mis padres Lydia y Josué,* que vivieron día a día y de principio a fin, cada uno de los momentos buenos y malos que se presentaron durante estos últimos años. Gracias mil por depositar en mí esa confianza que todos necesitamos para lograr nuestras metas, por sus sabios consejos que llevo siempre en mi corazón, los amo.

Y finalmente agradezco también a *Kika, Eli, Flor Isela, Javier* y *Román* por siempre estar al pendiente de mí en todo momento, por apoyarme cada uno en aspectos diferentes y echarme porras en aquellos momentos de flaqueza que sabían que no me iban a doblegar por mucho tiempo, los quiero mucho.

Resumen

En este trabajo se desarrolló un Sistema Cooperativo de Agentes Expertos (SiCAE), cuyos agentes son Sistemas Expertos (SE), basado en un modelo de cooperación, para la solución colectiva de problemas complejos.

El SiCAE se estructura de acuerdo al modelo organizacional *Aalaadin,* en base al rol que desempeña cada agente.

El modelo de cooperación que se propone, contempla un agente interfaz, para interactuar con el usuario, un agente administrador que se encarga de la coordinación, control e integración de resultados y una colección de agentes expertos en un dominio, pero con diferentes especialidades. En este modelo, todos los agentes se comunican entre sí.

Los agentes expertos tienen bases de conocimiento independientes, de acuerdo a su área de experiencia. El SiCAE se desarrolló utilizando la plataforma MultiAgent Development Kit (MadKit). Las bases de conocimiento de los agentes expertos se desarrollaron en Java Expert System Shell (Jess). Ambas plataformas se basan en el lenguaje de programación Java.

Para la integración de MadKit y Jess, se desarrolló una biblioteca de clases en Java, que permite la comunicación entre estas dos plataformas y la generación de agentes con diversos roles. Con dicha biblioteca se pueden generar agentes expertos en cualquier área de conocimiento.

La base de conocimiento de los agentes está estructurada en forma de un árbol de decisiones, que es procesado por el algoritmo Rete que utiliza Jess. Para la comunicación entre los agentes expertos, se utilizó el método expandAndAssert de la clase Rete de Jess. Con este método se pasa el parámetro current - que contiene el valor del nodo, es decir, la pregunta que se le hace al usuario - a la clase JessAgent la cual forma parte de la biblioteca desarrollada en este trabajo - a los otros agentes.

Como caso de estudio, se desarrolló un Sistema Cooperativo de Agentes Expertos en el área de Medicina, integrado por un médico internista y tres especialistas, en las áreas de Gastroenterología, Cardiología y Neurología.

Para probar el SiCAE, se planteó el problema general de diagnosticar una enfermedad de acuerdo a las respuestas dadas por el usuario y la cooperación entre los agentes expertos que integran el equipo. En particular, se utilizó el SiCAE para el diagnóstico de tres enfermedades.

Índice general

Capítulo 1

Introducción

1.1. Planteamiento

Los Sistemas Expertos (SE) abordan la solución de problemas de forma análoga a como lo haría un experto humano. Un experto humano necesita de conocimiento, razonamiento y experiencia para resolver un problema en un dominio de conocimiento específico, elementos que se tratan de implementar en un SE. Los elementos principales de un SE son una *base de conocimientos* que es la codificación del conocimiento proporcionado por el experto humano y la *máquina de inferencia* que hará una emulación del razonamiento humano, relacionando el conocimiento pertinente para obtener la solución esperada.

En la vida real es común que varios expertos humanos trabajen en equipo para lograr un objetivo común. Una analogía de estos equipos de trabajo en el área de las ciencias computacionales son los Sistemas Multiagentes (SMA), mediante los cuales se puede representar una sociedad de individuos con capacidad para seguir objetivos propios en base a la percepción del medio en el que se desenvuelven [Cuena, 1998].

Los SMA ofrecen un mecanismo para la solución distribuida de problemas. En un SMA cada agente tiene funciones y características específicas dentro de un ambiente que les permite colaborar y comunicarse con los demás agentes; para que juntos logren su objetivo común que es el solucionar un problema.

La tecnología de los SE y Agentes Inteligentes derivan de la Inteligencia Artificial (LA), una rama de las ciencias computacionales interesada en el diseño e implementación de programas, que sean capaces de emular las habilidades cognitivas humanas. Esta tecnología ha sido satisfactoriamente aplicada en diversos dominios. Otra rama de las ciencias computacionales que ha tenido gran interés por parte de los investigadores es la Inteligencia Artificial Distribuida (IAD), que se ha enfocado en los niveles de granularidad de la individualidad e

interacción, donde el objetivo es aplicar las visiones sociológicas, políticas y económicas para desarrollar lo que podría describirse más estrechamente como modelos computacionales de sociedades [Durfee, 1995].

La mayoría de las aplicaciones en SMA y SE, se han desarrollado de forma independiente. Así, los agentes o SMA realizan funciones asistenciales de gran ayuda para las personas dentro de una organización, como por ejemplo, agentes con características de asistentes personales. Los SE, por otro lado, son usados para la solución de un problema específico en algún dominio del conocimiento, como por ejemplo, SE para el diagnóstico de enfermedades.

En este trabajo se presenta una propuesta de integración de SMA y SE en un Sistema Cooperativo de Agentes Expertos (SiCAE), para la solución de un problema "complejo". La complejidad de un problema se determina por el tamaño y número de las bases de conocimientos, la incertidumbre de los datos y la presencia de información contradictoria que puedan proporcionar los expertos humanos.

Los sistemas colaborativos son aquellos que ayudan a un grupo de individuos a realizar una tarea común dentro de un ambiente común. Dentro de un grupo de trabajo se establecen roles a cada uno de los miembros para que el logro de los objetivos se lleve a cabo de una forma coordinada y más eficiente que si lo realizara un solo individuo.

En el Sistema Multiagentes Expertos realizado, los agentes son SE que tienen la capacidad de colaborar dentro de un equipo especializado, basándose en el conocimiento codificado en sus bases de conocimiento. Por lo tanto, el punto central de este trabajo es la Colaboración, la cual definiremos como la participación de individuos dentro de un grupo, en base a un conjunto de roles, mecanismos de percepción y reglas de colaboración, estos puntos se definen para este trabajo como:

- *Roles definidos,* son los privilegios y responsabilidades dentro del grupo.
- ***Reglas de colaboración,*** es la forma en la que cada individuo dentro del grupo cooperará con los demás.
- *Percepción,* es la forma en la que cada individuo percibe la información proporcionada para así dar su punto de vista a los demás.

Se espera que la solución que arroje la aplicación sea mejor al ser dada por un equipo de expertos en un dominio específico de conocimiento, que la solución proporcionada por cada SE de forma individual.

El diagrama de la Fig.1.1 muestra el planteamiento de conceptos y elementos a utilizar en este trabajo. El Sistema Multiagentes Expertos integra desde el punto de vista operacional a dos plataformas de desarrollo: Jess (Java Expert System Shell) para el desarrollo de SE y MadKit (MultiAgent Development Kit) para el desarrollo de SMA. Desde el punto de vista funcional, se tiene un equipo de agentes expertos, que mediante el paso de mensajes se logra un estado de cooperación entre ellos, llegando a una conclusión.

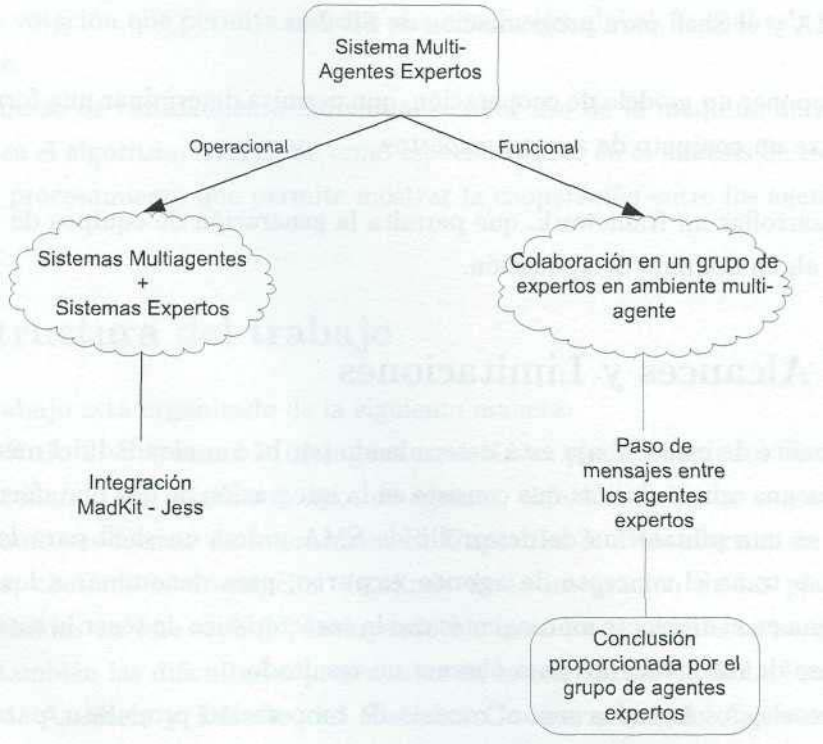

Figura 1.1: *Integración de elementos para el Sistema Cooperativo de Agentes Expertos (SiCAE).*

1.2. Objetivos

Objetivo general

Desarrollar un modelo de sistema multiagentes expertos cooperativos, para la solución de un problema "complejo".

Objetivos específicos

1. Aplicar el concepto de sistema multiexpertos para definir al SMA a desarrollar.

2. Complementar la documentación existente de la plataforma Jess, elaborando el diagrama de clases correspondiente.

3. Desarrollar un esquema de integración de la plataforma MadKit para el desarrollo de SMA y el Shell para programación de SE Jess.

4. Proponer un modelo de cooperación, que permita determinar una forma de interacción entre un conjunto de agentes expertos.

5. Desarrollar un framework, que permita la generación de equipos de agentes expertos, en algún dominio de aplicación.

1.3. Alcances y Limitaciones

El alcance de este trabajo está determinado por la complejidad del mismo. En esta tesis se plantea una aproximación, que consiste en la integración de dos plataformas de desarrollo: MadKit es una plataforma de desarrollo de SMA y Jess un shell para la creación de SE. De aquí se toma el concepto de agente experto, para denominar a los "individuos" que interactúan en el ambiente multiagentes con la característica de tener la capacidad de realizar un proceso de razonamiento para obtener un resultado.

Los resultados logrados son: el modelo de cooperación propuesto, para la integración e interacción de los agentes expertos; la adaptación de las herramientas computacionales que permitieron la implementación del framework para la generación de agentes expertos.

El modelo de cooperación propuesto se tomó como base para el desarrollo de la biblioteca de clases que genera equipos de agentes expertos, en un dominio de aplicación específico. Dicho modelo consta de tres partes principales:

1. Coordinación y Control del equipo de Agentes Expertos.

2. Asignación de tareas al equipo de Agentes Expertos.

3. Proceso de razonamiento e integración de resultados.

Sin embargo, en la etapa de desarrollo se hace una delimitación del modelo debido a que su alcance es muy amplio.

La tercera parte del modelo, que corresponde al proceso de razonamiento e integración de resultados, comprende una área de investigación que por sí sola se considera compleja. Por lo tanto, en este trabajo se delimita el desarrollo del modelo para representar las dos primeras partes y un proceso de razonamiento a nivel individual, ya que la integración de resultados implica el desarrollo de un mecanismo de mantenimiento del proceso de razonamiento y un mecanismo de votación que permita obtener una conclusión global, lo cual rebasa el alcance de este trabajo.

Para el proceso de razonamiento individual se hace uso de la máquina de inferencia de Jess, que utiliza el algoritmo RETE. Se tomó especial interés en el análisis de este algoritmo para lograr el procesamiento que permite mostrar la cooperación entre los agentes expertos activos.

1.4. Estructura del trabajo

El presente trabajo está organizado de la siguiente manera:

En el **Capítulo 2. Sistemas Multiagentes,** se da un panorama de los Sistemas Multi-agentes, mencionando sus orígenes y tipos de agentes, enfatizando en las organizaciones de agentes, así como sus formas de interacción. **El Capítulo 3. Plataforma de desarrollo MadKit-Jess,** contiene la descripción detallada de la integración de las plataformas de desarrollo utilizadas en este trabajo, incluyendo los modelos y arquitecturas involucrados. Se describen también las dificultades presentadas en la realización de dicha integración. En el **Capítulo 4. Análisis y Diseño del Sistema Cooperativo de Agentes Expertos (SiCAE),** se describe el análisis y diseño de la aplicación presentada en este trabajo. El **Capítulo 5. Caso de estudio** se detallan las pruebas realizadas en el SiCAE, analizando los resultados obtenidos. En el **Capítulo 6. Conclusiones y trabajo futuro,** se mencionan los resultados obtenidos en base a este trabajo de tesis, así como el trabajo futuro que se desprende del mismo.

Se incluyen dos apéndices: el **Apéndice A. Sistemas basados en conocimiento,** donde se da un panorama general de estos sistemas, debido a la relación con el trabajo propuesto. **El Apéndice B. Clases en Jess,** se muestran esquemas representativos de las aportaciones principales de este trabajo, como el Diagrama de Clases de Jess.

Capítulo 2

Sistemas Multiagentes

Junto con la Solución Distribuida de Problemas y la Inteligencia Artificial Paralela, los Agentes de Software y los Sistemas Multiagentes (MAS) forman colectivamente una de las áreas que se desprenden de la Inteligencia Artificial Distribuida (IAD) [Nwana, 1996].

El objetivo principal de la IAD es el estudio de modelos y técnicas para la resolución de problemas donde la distribución, ya sea física o funcional, sea inherente. En la IAD se utiliza la metáfora de la "inteligencia" que se fundamenta en diferentes metáforas de las ciencias exactas y sociales como la biología, la física y la sociología, entre otras. Se dice que los individuos heterogéneos e independientes del sistema son "inteligentes" si alcanzan un cierto grado de adaptación mutua [García y Ossowski, 1998].

2.1. Inteligencia Artificial Distribuida

La Inteligencia Artificial Distribuida (IAD), es una disciplina que se enfoca a desarrollar lo que podría describirse como modelos computacionales de sociedades [Durfee, 1995]. Las motivaciones, objetivos y beneficios potenciales principales de la IAD son [Nwana, 1996]:

- **Modularidad.** Permite reducir la complejidad.
- **Rapidez.** Debido al parelelismo.
- **Fiabilidad.** Debida a la redundancia.
- **Flexibilidad.** Las tareas se componen más fácilmente de una organización modular.

De forma general, los sistemas de IAD se caracterizan por tener una arquitectura formada por componentes inteligentes y modulares que interactúan de forma coordinada. Se pueden

mencionar algunas ventajas del enfoque de la IAD sobre los paradigmas convencionales [García y Ossowski, 1998]:

1. Así como los sistemas distribuidos convencionales aprovechan la distribución natural del dominio en espacio, tiempo y función, los sistemas de IAD lo hacen con el objetivo de mejorar el rendimiento, la robustez, facilitar reusabilidad y mantenimiento.

2. Los sistemas de IAD se pueden adaptar a estructuras preestablecidas en las organizaciones humanas y facilitan la interacción hombre-máquina.

3. Ayudan al desarrollo de modelos cognoscitivos de cooperación y coordinación y se apegan a teorías lengüísticas, psicológicas, sociológicas y filosóficas.

Una de las ramas de la IAD, se refiere a los Sistemas Multiagentes (SMA). Los SMA tienen como objetivo proporcionar dos principios de construcción de sistemas complejos, que son: l)múltiples agentes y 2)mecanismos para la coordinación de comportamientos de los agentes independientes [Stone y Veloso, 1997]. En este enfoque, la "inteligencia" se obtiene como resultado de la interacción de elementos simples. En este trabajo se utiliza un SMA para modelar la cooperación entre expertos, a fin de obtener una mejor solución a problemas complejos.

2.1.1. Clasificación Colectiva

Con la finalidad de dar solución a problemas complejos que surgen en ciencias como la Medicina, la Geología y la Biología, entre otras, existen métodos de clasificación que se basan en la Estadística, las Redes Neuronales Artificiales, el Análisis Sintáctico Estructural, el Análisis Lógico-Combinatorio, etc. Sin embargo se considera que estos métodos no dan solución de forma individual a los problemas, sino que ofrecen una eficiencia de solución mayor de forma combinada [Riquenes y Alba, 1997].

Los modelos de clasificación colectiva explotan las dependencias en una red de objetos para mejorar las predicciones. Un modelo relacional capaz de expresar y razonar con sus dependencias puede alcanzar un desempeño superior a los modelos relaciónales que ingnoran sus dependencias. Como ejemplo se tienen a las Redes de Dependencia Relacional (RDN), que son modelos de clasificación colectiva que ofrecen un mejor rendimiento comparadas con árboles de probabilidad relacional [Neville y Jensen, 2003].

A pesar de que la Clasificación Colectiva es una línea de investigación poco explorada, se tiene la hipótesis de que el comportamiento colectivo de un grupo de clasificadores sobre

un objeto, proporciona información adicional que no se da con soluciones de un clasificador individual. Este conocimiento adicional puede proporcionar resultados más precisos y consistentes para la solución de un problema complejo [Riquenes y Alba, 1997].

En los trabajos encontrados en referencia a Clasificación Colectiva, se hacen uso de diversos métodos de clasificación como: Votación por Simple Mayoría, Votación Pesada Globalmente, Votación Pesada por Objetos, Clasificaicón según Preselección y las Redes de Dependencia Relacional, dando como resultado un mayor nivel de eficiencia sobre las aproximaciones de clasificación no colectiva.

Un ejemplo de problema donde se utiliza una combinación de técnicas de Inteligencia Artificial (IA) para su solución, son los sistemas financieros, donde el modelado de los procesos consiste en trazar un conjunto de series de tiempo de sus variables principales dentro de una descripción típicamente continua, la cual se espera que dé mas información que los datos originales. Sin embargo, los métodos estándar para el análisis de series de tiempo no son capaces de tratar con comportamientos no lineares y estacionarios. Por lo que el uso de diversas técnicas de IA para modelar sistemas dinámicos han dado buenos resultados. Dentro de estas técnicas se tienen: Redes Neuronales Artificiales, Lógica Difusa, Razonamiento Basado en Casos, Algoritmos Genéticos y Sistemas Multiagentes [Soto y Núñez, 2003].

Se considera que este tipo de clasificación puede ayudar para determinar un método apropiado para que los Agentes Expertos planteados en este trabajo de tesis obtengan un resultado aceptable como conclusión global. Esto da la posibilidad de continuar con trabajos. de investigación que puedan complementar la información aportada en este trabajo.

2.2. Agentes de Software

Con el nombre de agentes, han proliferado una gran variedad de programas. Se puede decir que ha habido una explotación excesiva de este término, sin el correspondiente consenso de su significado. Entre las definiciones de agente que se encuentran frecuentemente en la literatura están las siguientes:

Weiss define a un agente como una entidad computacional, como un programa de software o un robot, que puede ser visto como un ente dotado de perceptores y actuadores en un ambiente y que es autónomo en su comportamiento [Weiss, 1999].

Stuart Russel [Russel y Norvig, 1995], define a un agente como una entidad que percibe su entorno a través de sensores y actúa sobre ese entorno a través de efectores o permanece como una entidad que presenta características de la inteligencia humana, trabajando de una

forma autónoma y continua en su ambiente. Un agente es racional cuando realiza la mejor acción posible considerando sus objetivos y metas.

Pattie Maes [Maes, 1997], del Software Agents Group del MIT Media Laboratory, define a un agente como un sistema computacional que tiene una larga vida, objetivos, sensores y efectores y decide autónomamente qué acciones realizar en la situación actual para maximizar el avance hacia sus objetivos (cambiantes en el tiempo).

Walter Brenner [Brenner, Zarnekiq y Wittig, 1998], define a un agente de software inteligente como un programa de software que puede realizar tareas específicas para un usuario y posee un grado de inteligencia suficiente para ejecutar parte de sus tareas de forma autónoma y para interactuar con su entorno de forma útil.

Es difícil escoger una definición, ya que cada autor tiene un punto de vista de acuerdo a su línea de investigación. En este trabajo se toma una combinación de la definición de Stuart Russel y Pattie Maes: de acuerdo al caso de estudio que se presenta, se puede decir que un agente es una entidad que percibe su entorno y actúa sobre él, tomando decisiones en cuanto a las acciones a realizar para alcanzar sus objetivos.

Desde un punto de vista descriptivo, se distinguen dos usos comunes para el término de agente: la noción blanda y la noción dura [Wooldridge, 1995]. A continuación se describen las características que engloban cada una de las nociones mencionadas.

2.2.1. Noción blanda de agencia

Quizás, la forma más general en la cual el término de agente es utilizado para denotar hardware o (más usualmente) sistemas computacionales basados en software, es aquel que tiene las siguientes propiedades [Wooldridge, 1995]: autonomía, habilidad social, reactividad, proactividad, orientación a objetos y continuidad temporal.

Una forma simple de conceptualizar un agente dentro de esta noción blanda, es como un demonio de software (daemons). Este tipo de procesos tienen las características mínimas para ser considerados como agentes, es decir, tienen un cierto grado de autonomía, reactividad y habilidad comunicativa.

Esta noción blanda de agencia, es típicamente usada en la Ingeniería de Software basada en Agentes.

2.2.2. Noción dura de agencia

Para algunos investigadores - particularmente los que trabajan en IA - el término "Agente" tiene un significado más estricto que el descrito arriba. Estos investigadores generalmente designan un sistema computacional como agente, si además de tener las propiedades arriba mencionadas, es descrito utilizando conceptos que son aplicados más usualmente al comportamiento humano, como son [Wooldridge, 1995]: movilidad, veracidad, benevolencia, racionalidad, adaptabilidad y colaboración.

De acuerdo a esta definición, un proceso demonio de Unix no puede considerarse como un agente.

En este trabajo se puntualiza el término de agentes expertos a los individuos participantes dentro de un ambiente, por lo que de acuerdo a sus características se puede decir que tales agentes se clasifican dentro de la noción dura arriba mencionada. Los agentes expertos tienen características del comportamiento humano como la racionalidad y la colaboración, es decir, no se considera que solo sean agentes de software.

2.3. Agentes Inteligentes

Uno de los objetivos principales de la Inteligencia Artificial (IA), es la programación de computadoras para que realicen tareas que actualmente son hechas mejor por los humanos, ya que involucran procesos mentales de alto nivel tales como aprendizaje perceptivo, organización de memoria y razonamiento juicioso [Minsky, 1968].

Los agentes presentan características adecuadas para modelar aspectos del comportamiento inteligente como: autonomía, proactividad y colaboración, por lo que son una posibilidad viable para la construcción de sistemas inteligentes, capaces de emular el comportamiento humano.

2.4. Espacio de Agentes

Para proporcionar una forma más sencilla de caracterizar el espacio de los tipos de agentes que pueden resultar al tratar de describir las posibles combinaciones de sus atributos, varios investigadores de la comunidad de agentes, han propuesto esquemas y taxonomías de clasificación.

Moulin y Chaib-draa caracterizan a los agentes por su grado de capacidad para la solución de problemas: [Bradshaw, 1999] "Un *agente reactivo,* reacciona a cambios en su ambiente o

a mensajes de otros agentes... Un *agente intencional* es capaz de razonar sus intenciones y deseos, para crear planes y acciones, y así ejecutarlos. Adicionalmente a las capacidades de los agentes intencionales, un *agente social* posee modelos explícitos de otros agentes" [Moulin y Chaib-draa, 1996].

Gilbert (1995) describe a los *Agentes Inteligentes* en términos de un espacio definido por tres dimensiones: *agencia, inteligencia y movilidad.*

- **Agencia:** Esta dimensión es considerada como el grado de autonomía y autoridad que tiene un agente.

- **Inteligencia:** Es considerada como el grado de razonamiento y de comportamiento aprendido.

- **Movilidad:** Este término hace referencia a la capacidad que tienen los agentes para ejecutarse en un ambiente distribuido.

El diagrama de la Fig.2.1 proporciona una clasificación de los *Agentes Expertos* dentro de este espacio.

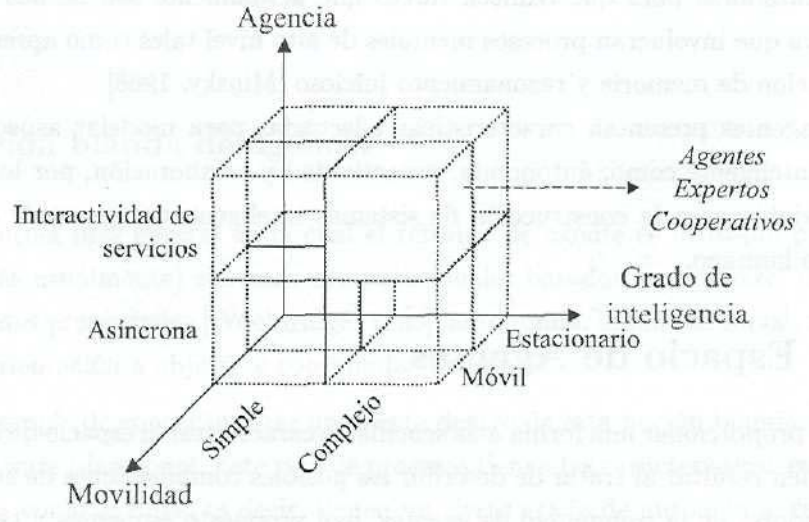

Figura 2.1: *Alcances de los Agentes Expertos.*

2.5. Interacción en Sistemas Multiagentes

En un SMA, se considera que los agentes están dentro de una sociedad y necesitan *interactuar* o *comunicarse* con los demás agentes. Los agentes pueden comunicarse de diversas formas, de acuerdo a su comportamiento y actividades, como se muestra en la Fig.2.2.

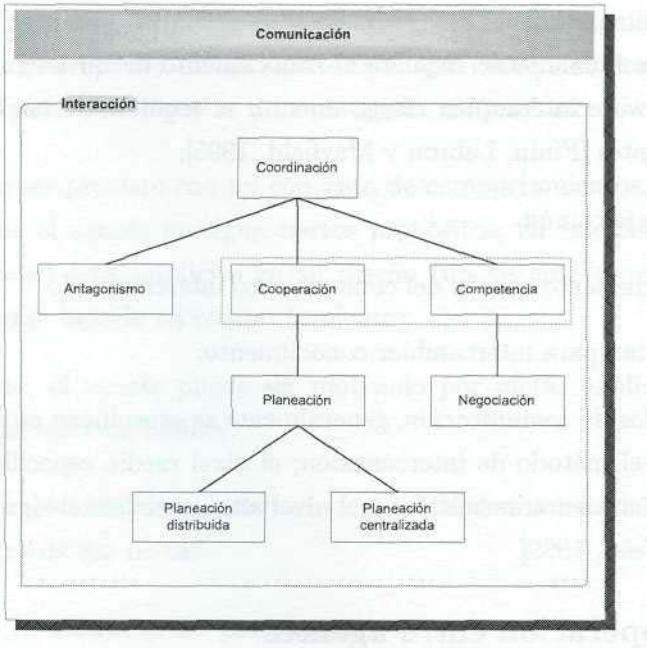

Figura 2.2: *Esquema de comunicación entre agentes*

En la figura 2.2 se puede observar como forma general de comunicación a la *interacción,* que es todo lo que ocurre entre los agentes y su ambiente. Los agentes pueden interactuar de forma directa a través de algún tipo de comunicación (intercambiando información), o de una forma indirecta por medio de su ambiente (factores que modifican el estado de su ambiente). Las formas de interacción que juegan el papel más importante dentro de la IAD son la *cooperación y* la *competencia* [Weiss, 1999], esta conexión se representa con una línea punteada en el diagrama.

Un componente fundamental de la interacción es la *coordinación,* que es la propiedad de un sistema de agentes para realizar alguna actividad en un ambiente compartido, reduciendo la disputa de recursos, evitando interbloqueos y manteniendo condiciones de seguridad aplicables [Weiss, 1999]. De la coordinación se desprende el *antagonismo;* la *cooperación,* que es la coordinación entre agentes *no antagónicos;* la *competencia,* que es la movilización de

diversos recursos cognitivos para enfrentar algún tipo de situaciones; la *negociación,* que es la coordinación entre agentes competitivos o auto-interesados. Para tener una cooperación satisfactoria, cada agente debe mantener un modelo de los otros agentes, así como también desarrollar un modelo de futuras interacciones, lo que presupone sociabilidad [Weiss, 1999].

Como último nivel en el diagrama, se tiene a la *planeación* que depende de la *cooperación* y la *negociación.*

Como en los humanos se requiere el conocimiento de un lenguaje común, para que los agentes de software interactúen efectivamente, se requiere de las tres componentes fundamentales siguientes [Finin, Labrou y Mayfield, 1995]:

1. Un lenguaje común
2. Un entendimiento común del conocimiento intercambiado
3. La habilidad para intercambiar conocimiento.

Los *protocolos de comunicación,* generalmente se especifican en tres niveles. El nivel más bajo especifica el método de interconexión; el nivel medio especifica el formato, o sintaxis de la información a ser transmitida; y el nivel alto especifica el significado o semántica de la información [Weiss, 1999].

2.5.1. Cooperación entre agentes

La aproximación de SMA para modelado, aparece como un *framework* apropiado para abordar la representación y coordinación de soluciones múltiples como la integración de múltiples puntos de vista.

Por naturaleza, la solución de problemas no es un proceso lineal, por otra parte, el comportamiento humano no siempre es consistente y lineal durante el proceso de solución de problemas. Los SMA capturan este tipo de análisis difuso, debido a que están diseñados para soportar la concurrencia y el no-determinismo [Marcenac, Leman y Giroux, 1996].

La forma en que las personas realizamos nuestros proyectos es el trabajo en común, por ello es lógico que la resolución de un problema complejo se tienda a abordar de forma cooperativa. Por ejemplo en un ambiente académico se necesitan ciertas relaciones de cooperación como: alumno-profesor, alumno-alumno o profesor-profesor, para alcanzar los objetivos dentro del grupo [Ortega et al., 1997].

Para cooperar eficientemente con sus compañeros, un agente debe representar cualquier estructura social en la que juegue un papel, y también *razone* con esa representación. Una

estructura social es un conjunto de relaciones que se sostienen entre los agentes de una sociedad [d'Inverno, Luck y Wookdridge, 1997].

El concepto de cooperación puede tener diferentes variaciones, hay autores que manejan diferentes puntos de vista. Por ejemplo Norman [Doran, Franklin y Jennings, 1997], define a la cooperación de la siguiente forma: "Cooperar es actuar con otro u otros para un propósito y beneficios comunes". De esta definición es necesario entender qué significa "actuar para un propósito en común". El *propósito* de un agente es el que maneja su *comportamiento,* pero hay dos formas principales en las que se da a un agente un *propósito:*

- El agente puede ser provisto con un conjunto de comportamientos, que son diseñados de tal forma que el agente persigue ciertos propósitos; en este sistema de control el propósito (o meta) está implícito en su diseño (un sistema de control orientado a metas o puramente basado en comportamiento).
- Alternativamente, el agente puede ser motivado por metas explícitas, posiblemente derivadas de más motivos básicos.

Un agente puede emplear la planeación y otros procesos de toma de decisiones para dirigir su acción hacia el logro de sus metas.

La cooperación en la solución de problemas

La cooperación involucra un conjunto de agentes que interactúan mediante la comunicación de información entre ellos mientras se soluciona el problema. Los agentes pueden ser agregados al conjunto como un comité, o pueden estar más formalmente organizados como una jerarquía. La información intercambiada entre los agentes puede ser incorrecta, por lo que es posible que se altere el comportamiento de los agentes que reciben dicha información [Clearwater, Hogg y Huberman, 1992].

Por tanto, la cooperación es básicamente el proceso de distribuir objetivos, planes y tareas entre los diversos agentes existentes en el ambiente [Haugeneder y Steiner, 1998].

2.6. Comunicación y representación del conocimiento en agentes

Al hablar de un agente, se asume que tiene conocimiento representado de forma explícita y un mecanismo para realizar inferencias con ese conocimiento. También se asume que un

agente tiene la *habilidad de comunicación,* para alcanzar sus objetivos en dos niveles: 1) a nivel propio y 2) a nivel de la sociedad en la que existen [Weiss, 1999]. La *comunicación* entre agentes se puede dar por medios propios del ambiente de desarrollo de SMA a través de lenguajes de comunicación estándar [Gilbert, 1997].

2.6.1. Ontologías y conocimiento

El conocimiento representado formalmente se basa en una conceptualización de objetos y otras entidades que existen en algún área de interés y las relaciones que las mantienen. Cada base de conocimiento, sistema basado en conocimiento o agente con nivel de conocimiento está comprometido con alguna conceptualización, ya sea explícita o implícita [Gruber, 1993].

Una o*ntología* es una especificación explícita de una conceptualización. Cuando el conocimiento de un dominio es representado en un formalistmo declarativo, el conjunto de objetos que pueden ser representados es llamado el universo del discurso. Este conjunto de objetos y las relaciones entre ellos, se reflejan en un vocabulario representativo con el cual un programa basado en conocimiento representa el conocimiento [Gruber, 1993].

Los SBC poseen requerimientos especiales para su interoperabilidad, ya que estos sistemas operan y se comunican utilizando declaraciones en una representación del conocimiento formal. Los agentes en un ambiente de IAD, interactúan para intercambiar conocimiento. Para tal comunicación de nivel de conocimiento, se necesitan comunicaciones en tres niveles: *formato de representación del lenguaje, protocolo de comunicación de agentes,* y *especificación del contenido del conocimiento compartido* [Gruber, 1993].

Las propuestas para los formatos de representación de conocimiento estándar y los lenguajes de comunicación de agentes son independientes del contenido de conocimiento que es intercambiado o comunicado. Las ontologías pueden ser utilizadas para convenciones del tercer tipo: *especificaciones de contenido específico* [Gruber, 1993].

2.6.2. Lenguajes de comunicación

Lenguaje de manipulación y consulta de conocimiento (KQML)

La comunicación humana en forma oral, se utiliza como modelo para la comunicación entre los agentes, en donde la teoría de actos del habla (speech act theory) es la base. Un acto del habla consta de tres aspectos: locución, que es la pronunciación física del locutor; ilocución, que es el significado intencional de la pronunciación del locutor y perlocución, que es la acción que resulta de la locución.

La teoría de actos del habla utiliza el término *performative* para identificar la fuerza de ilocución en determinada pronunciación. Algunos ejemplos de *performatives* son: *promise, report, convince, insist, tell, request* y *demand.* La teoría de actos del habla ayuda a definir el tipo de mensaje utilizando el concepto de fuerza de ilocución [Weiss, 1999].

KQML utiliza *performatives* que son modeladas sobre actos del habla, de esta forma la semántica de las *performatives* son de dominio independiente, mientras que la semántica del mensaje está definida por una serie de campos como: contenido, lenguaje y ontología, entre otros parámetros [Weiss, 1999]. Este lenguaje de manipulación y consulta de conocimiento es utilizado para comunicar actitudes acerca de la información, tales como consultas, declaraciones, creencias, requerimientos, alcances, suscripciones y ofertas [Finin y Weber, 1994].

KQML es considerado tanto un formato de mensajes como un protocolo para el manejo de mensajes, que soporta la compartición de conocimiento entre agentes dentro de la solución distribuida de problemas. KQML se enfoca en un conjunto de *performatives,* que definen las operaciones permitidas que los agentes pueden intentar en el conocimiento y objetivos de los otros agentes [Patil et al., 1992]. Las expresiones de este lenguaje consisten del contenido de una expresión encapsulada en una envoltura de mensaje el cual a su vez está encapsulada en una envoltura de comunicación.

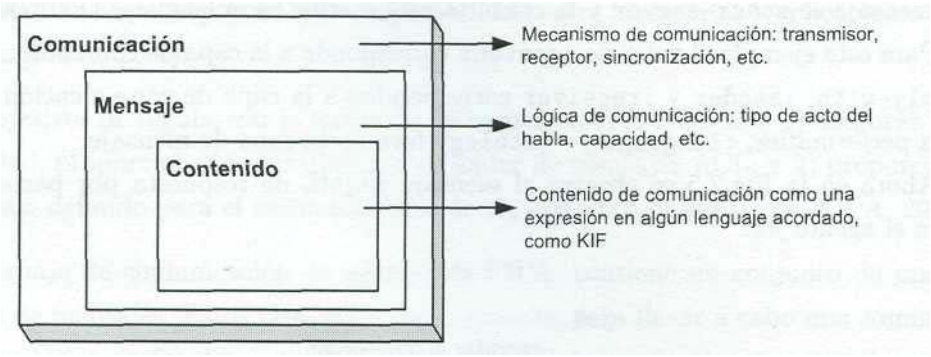

Figura 2.3: *Expresiones en KQML*

De acuerdo al diagrama representado en la Fig.2.3, se tienen tres capas: contenido, mensaje y comunicación. La capa de contenido incluye una expresión en algún lenguaje que codifica el conocimiento a ser transmitido. El propósito principal de la capa de mensaje es identificar el acto del habla o *performative* que el transmisor adjunta al contenido, puede ser una aserción, una pregunta o un comando, y cualquier conjunto de calificadores que puedan ser apropiados para la *performative.* La capa final de comunicación agrega una se-

gunda capa de características del mensaje que describen los parámetros del nivel bajo de comunicación, tales como la identidad, transmisor y receptor, un identificador único asociado con la comunicación y si la comunicación es síncrona o asíncrona [Patil et al., 1992].

En la Fig.2.4 se muestra el ejemplo de un mensaje KQML que un agente llamado *joe* envía a un servidor para hacer una consulta sobre una parte del inventario de IBM:

```
(ask-one
        :sender joe
        :content (PRICE IBM ?price)
        receiver stock-server
        :reply-with ibm-stock
        :language LPROLOG
        :ontology NYSE-TICKS)
```

Figura 2.4: *Mensaje KQML de consulta.*

En este mensaje la *performative* KQML es ask-one, el contenido es (PRICE IBM ?price), la ontología aceptada por la consulta es identificada por el token nyse-ticks, el receptor del mensaje es stock-server y la consulta está escrita en el lenguaje LPROLOG.

Para este ejemplo el valor de : content corresponde a la capa de contenido; los valores de :reply-with, :sender y :receiver corresponden a la capa de comunicación y, el nombre de la *performative,* :language y :ontology forman la capa de mensaje.

Ahora en la Fig.2.5 se observa el mensaje KQML de respuesta por parte del servidor hacia el agente *joe.*

```
(tell
        :sender stock-server
        :content (PRICE IBM 14)
        receiver joe
        :in-reply-to ibm-stock
        :language LPROLOG
        :ontology NYSE-TICKS)
```

Figura 2.5: *Mensaje KQML de respuesta.*

Formato de intercambio de conocimiento (KIF)

El Knowledge-Sharing Effort (KSE), es una iniciativa para el desarrollo de una infraestructura técnica de comunicación que soporte la distribución de conocimiento entre sistemas [Finin, Labrou y Mayfield, 1995].

KIF es la solución sugerida por el KSE para los aspectos sintácticos de la representación de conocimiento compartido. Este lenguaje es propuesto como un vehículo para expresar conocimiento y meta-conocimiento.

KIF es un lenguaje lógico que ha sido propuesto como un estándar para describir cosas dentro de los sistemas expertos, bases de datos, agentes inteligentes, etc. Por otra parte, está diseñado específicamente para servir como mediador en la traducción de otros lenguajes [Weiss, 1999].

Lenguaje de Comunicación de Agentes (FIPA-ACL)

La Fundación para Agentes Inteligentes Físicos (FIPA), es una organización internacional dedicada a promover la industria de los agentes inteligentes mediante el desarrollo abierto de especificaciones que soporten la interoperabilidad entre los agentes y aplicaciones basadas en agentes [FIPA, 2000].

Los objetivos de regularizar la forma de un mensaje FIPA-ACL son: 1) asegurar la interoperabilidad, proporcionando un conjunto estándar de mensajes ACL, y 2) proporcionar un proceso bien definido para el mantenimiento de ese conjunto de mensajes [FIPA, 2000].

El lenguaje de comunicación de agentes de FIPA, contiene un conjunto de uno o más elementos de mensajes. Estos elementos son necesarios para llevar a cabo una comunicación efectiva y pueden variar de acuerdo a la situación. El único elemento que es obligatorio en todos los mensajes ACL es la *performative,* aunque se espera que la mayoría de los mensajes también contengan los elementos: transmisor, receptor y contenido.

La perspectiva de interoperabilidad en los ambientes computacionales actuales, han sido fundados por el consorcio de la aproximación del Knowledge Sharing Effort (KSE). ACL es una implementación de KQML que difiere del KQML puro en cuanto a que hace que KIF sea el lenguaje de contenido de las aplicaciones en interacción. Una de las soluciones de comunicación sugeridas por el KSE es el lenguaje de comunicación KQML (Knowledge Query and Manipulation Language) [Finin, Labrou y Mayfield, 1995].

2.7. Herramientas para el desarrollo de Sistemas Multi-agentes

Existen diversas herramientas o entornos de desarrollo de Sistemas MultiAgentes (SMA). En esta sección se dará un panorama de las herramientas más importantes.

Dentro de estos entornos se pueden encontrar lenguajes, armazones y metodologías para el desarrollo de SMA. Los lenguajes parten de principios que se basan en modelos operacionales y formales de los SMA, los armazones son aquellos que permiten construir un SMA a través de una plataforma que proporciona servicios de administración de agentes, comunicación y una arquitectura de agentes. Pero para construir un SMA ya sea con un lenguaje o con un armazón, es necesario tener una metodología que es la que establece la forma en la que se va a construir el SMA [Pavón y Gómez, 2003].

2.7.1. Lenguajes de agentes

Entre los lenguajes más conocidos están **Agento** y **ConGOLOG. AgentO** [Shoham, 1993], es considerado un paradigma de programación basado en una visión social de la computación, llamado Programación Orientada a Agentes. En este lenguaje un agente es la entidad principal, cuyo estado se ve como un conjunto de componentes mentales (creencias, habilidades, elecciones y compromisos) [Pavón y Gómez, 2003]. El componente que determina cómo actúa el agente es el conjunto de reglas compromiso [Weiss, 1999]. Por otra parte **ConGOLOG** [Lespérance et al., 1996], es un lenguaje de agentes que se basa en teorías de agentes y depende de métodos de demostración automática, por lo que a diferencia de AgentO, sí requiere del uso de la Lógica Matemática. En ConGOLOG se modela la ejecución de tareas asignadas a varios agentes y su efecto en el entorno en el que están actuando, por esta razón se tienen que predecir los efectos de las acciones sobre el entorno, lo que algunas veces resulta imposible [Pavón y Gómez, 2003].

Las desventajas de este tipo de lenguajes de agentes es que a diferencia de lenguajes convencionales como Java o C++, no ofrecen mecanismos de abstracción y encapsulación, lo que limita el desarrollo de sistemas con cierta complejidad. Con estas limitantes de los lenguajes de agentes, actualmente es más común utilizar plataformas y armazones de desarrollo de SMA [Pavón y Gómez, 2003].

2.7.2. Plataformas de desarrollo de SMA

Dentro de las plataformas de desarrollo más comunes están **JADE (Java Agent DEvelopment Framework) y Grasshopper.** JADE es una estructura que está implementada completamente en Java y es la implementación oficial del estándar FIPA, y soporta todos los servicios de infraestructura especificados en FIPA como: comunicaciones, movilidad, gestión de agentes y localización de agentes [Pavón y Gómez, 2003]. Esta plataforma puede ser distribuida a través de varias máquinas, las cuales incluso no necesitan compartir el mismo Sistema Operativo y la configuración puede ser controlada vía un GUI remoto para controlar y monitorear los estados de los agentes. Se tiene una solución multihilo para la implementación de los agentes, ya que cada uno de ellos es considerado una hebra, además de soportar una programación de comportamientos cooperativos. Permite también una integración completa con JESS (Java Expert System Shell), en donde JADE proporciona el Shell para el agente y Jess es la máquina de inferencia del agente para realizar el razonamiento necesario [Bellifemine y Trueco, 2001].

Por otra parte Grasshopper es una plataforma de agentes móviles que se construye sobre un ambiente de procesamiento distribuido [IKV++, 2001]. De esta forma se puede lograr una integración del paradigma tradicional cliente/servidor y la tecnología de agentes móviles. Esta plataforma es desarrollada conforme al primer estándar de agentes móviles de OMG (Object Management Group), es decir, MASIF (Mobile Agent System Interoperability Facility). El estándar MASIF se inició para lograr la interoperabilidad entre plataformas de agentes móviles de diferentes fabricantes. Una de las características principales de esta plataforma es que cuenta con una estructura llamada *Ambiente de agentes distribuidos,* que está compuesta de regiones, lugares agencias y dos tipos de agentes que son: agentes estacionarios y agentes móviles. La ventaja que tiene esta estructura está en los servicios que proporciona como: comunicación, registro, administración, transporte, seguridad y persistencia [IKV++, 2001].

Una plataforma de desarrollo de SMA que ofrece características convenientes para la realización de este trabajo es MadKit (Multi-Agent Development Kit), ya que consta de un modelo organizacional establecido y facilidades generales de un agente como: administración del ciclo de vida, paso de mensajes, distribución, etc. y permite una alta heterogeneidad en las arquitecturas de agentes y lenguajes de comunicación. Esta plataforma se explicará ampliamente en el siguiente capítulo.

2.7.3. Armazones de desarrollo de SMA

Dentro de los armazones más conocidos para el desarrollo de SMA, se encuentra ZEUS. Sus características principales son que presenta un entorno de desarrollo visual y consta de una herramienta y una metodología para el desarrollo de SMA. Entre otros aspectos, ZEUS facilita el desarrollo de aspectos como la coordinación, aunque no deja muy claras decisiones importantes como: ¿Qué hay que coordinar y para qué?, tiene la capacidad de combinar en los agentes: planificación, ontologías, asignación de responsabilidades, relaciones sociales, poniendo esto en un sistema funcional o herramienta [Pavón y Gómez, 2003].

La metodología que ZEUS propone, consta de 4 etapas que se mencionan a continuación [Pavón y Gómez, 2003]:

1. Análisis del dominio
2. Diseño de los agentes
3. Realización de los agentes
4. Soporte en tiempo de ejecución

Estas etapas de la metodología hacen uso de roles para analizar el dominio y para la asignación de los agentes.

Sin embargo, ZEUS tiene la desventaja de que hace complejo el diseño tanto como los lenguajes de agentes, debido a las diversas posibilidades de configuración de un SMA a través de su entorno: se tienen que suministrar una ontología, reglas de comportamiento, planes de ejecución de tareas y mensajes a enviar a otros agentes [Pavón y Gómez, 2003].

2.7.4. Metodologías para el desarrollo de SMA

Las metodologías orientadas a agentes son técnicas de análisis y diseño de sistemas de agentes, que han surgido como extensiones de las metodologías existentes orientadas a objetos y de metodologías de ingeniería del conocimiento. Son importantes para el paradigma de la programación orientada a agentes, ya que permiten la generalización en el desarrollo de aplicaciones orientadas a agentes [Iglesias, Garijo y González, 1998].

La utilización de extensiones de metodologías orientadas a objetos para la construcción de sistemas basados en agentes se da debido a las semejanzas que hay entre los paradigmas orientado a objetos y orientado a agentes. Estos paradigmas incluyen el paso de mensajes

como medio de comunicación, la diferencia está en que los mensajes entre agentes son predeterminados, además de que se considera que un agente tiene un *estado mental* que está compuesto por sus creencias, intenciones, deseos, acuerdos, etc. De igual forma, son utilizados los lenguajes orientados a objetos para el desarrollo de sistemas basados en agentes debido a su empleo habitual y entorno natural de desarrollo [Iglesias, Garijo y González, 1998].

A continuación se dará una introducción a las metodologías de agentes más conocidas.

Dentro de las extensiones de metodologías orientadas a objetos detalladas por Iglesias [Iglesias, Garijo y González, 1998], se encuentran:

- La metodología de análisis y diseño orientada a agentes de Burmeister.
- La técnica de modelado de agentes BDI.
- El método basado en escenarios multiagentes (MASB).
- La metodología orientada a agentes para modelado de empresas.

La primera sugiere tres modelos para analizar un sistema de agentes: modelo de agente, modelo de organización y modelo de cooperación. La segunda define dos vistas para modelar a los agentes BDI, la vista externa que consiste en descomponer el sistema en agentes y definir sus interacciones a través de los modelos de agentes e interacción respectivamente. La vista interna modela las clases de agentes BDI mediante tres modelos que son: modelo de creencias, modelo de objetivos y modelo de planificación. La tercer metodología propone desarrollar SMA en el área del trabajo cooperativo, y consta de las fases de análisis y diseño. El análisis tiene cuatro etapas: descripción de escenarios, descripción funcional de los papeles, modelado conceptual de los datos y del mundo, y modelado de la interacción sistema-usuario. El diseño consta de: descripción de los escenarios y de la arquitectura del SMA, modelado de objetos, modelado de agentes y modelado de conversaciones y validación global del sistema, aunque éste último paso solamente se propone. La última metodología propone una combinación de metodologías orientadas a objetos y metodologías de modelado de empresas IDEF (Integration DEfinition for Function modelling). Los modelos que propone son: modelo de funciones, modelo de casos de uso, modelo dinámico y sistema orientado a agentes que a su vez está compuesto de: identificación de agentes, protocolos de coordinación, invocación de planes y creencias, sensores y actuadores [Iglesias, Garijo y González, 1998].

Como se mencionó antes, existen extensiones de metodologías de ingeniería del conocimiento que sirven de base para modelar sistemas basados en agentes. La razón de utilizar

estas metodologías es porque los agentes tienen características cognitivas y el hecho de definirle conocimiento se considera un *proceso de adquisición de conocimiento*. La desventaja que existe en las metodologías de ingeniería del conocimiento es que consideran centralizado a un sistema basado en conocimiento, por esta razón no contemplan aspectos de agentes como la conducta, socialización o distribución [Iglesias, Garijo y González, 1998].

Algunas extensiones de estas metodologías son: CoMoMAS (Contribution to Knowledge Acquisition and Modelling in a Multi-Agent Framework) y MAS-CommonKADS. CoMoMAS es una metodología que extiende de CommonKADS, esta última se basa en un modelo de experiencia, es decir, para desarrollar sistemas expertos que interactúan con el usuario considerando dos agentes básicos: el sistema y el usuario. Por lo tanto su modelo de comunicación maneja interacciones hombre-máquina [Pavón y Gómez, 2003]. Continuando con CoMoMAS, es una metodología que modela SMA a través de los siguientes modelos: modelo de agentes; modelo de experiencia que se subdivide en el conocimiento de tareas, conocimiento de resolución de problemas y el conocimiento reactivo; modelo de tareas; modelo de cooperación; modelo del sistema y modelo del diseño [Iglesias, Garijo y González, 1998]. La segunda metodología MAS-CommonKADS, también extiende los modelos de CommonKADS, sólo que adiciona técnicas de las metodologías de ingeniería de protocolos y orientadas a objetos para establecer los protocolos entre los agentes. Esta metodología se basa en un ciclo de vida en espiral e incluye el análisis y diseño. Los modelos para el análisis son: modelo de agente, modelo de tareas, modelo de experiencia, modelo de organización, modelo de coordinación y modelo de comunicación. Los modelos del diseño son: diseño de la red, diseño de los agentes y diseño de la plataforma [Iglesias, Garijo y González, 1998].

Las metodologías orientadas a agentes son necesarias para definir el curso del análisis y diseño para el desarrollo de sistemas basados en agentes, pero no hay ningún estándar que defina cuándo hay que utilizar alguna en específico. Por ejemplo, aunque MAS-CommonKADS ofrece una estructura organizativa, no toma a la organización como una entidad, como lo hace el modelo *Aalaadin* que proporciona *estructuras de grupo* formadas por las instancias de grupo, agente y rol. Este modelo pertenece a la plataforma de desarrollo MadKit que se describe en la sección siguiente.

Capítulo 3

Plataforma de desarrollo MadKit - Jess

En este capítulo se presentan detalladamente las plataformas de desarrollo utilizadas en este trabajo. Como ya se ha mencionado, MadKit y Jess cuentan con las características necesarias para que se puedan integrar, ofreciendo así una plataforma para el desarrollo de un conjunto de agentes expertos colaborando en la solución de un problema.

3.1. MadKit (Multi-Agent Development Kit)

MadKit es una plataforma genérica para el desarrollo de Sistemas Multiagentes escrita en Java y construida bajo un modelo organizacional. Proporciona recursos generales a los agentes como: administración del ciclo de vida, paso de mensajes, distribución de tareas, entre otros. Además, permite una alta heterogeneidad en arquitecturas de agentes y lenguajes de comunicación [Gutknecht y Ferber, 1997].

La plataforma MadKit se está desarrollando en la Universidad de Montpellier, en el Laboratorio de Informática, Robótica y Micro-Electrónica. Se ha decidido utilizarla para este trabajo por las características de su modelo organizacional *(estructuras de grupo)* y la facilidad de ser una plataforma basada completamente en Java, permitiendo la compatibilidad con programas independientes basados también en Java [Gutknecht y Ferber, 1997].

3.1.1. Modelo Conceptual de MadKit

La principal ventaja de MadKit es una arquitectura basada en los conceptos de *Agentes, Grupos* y *Roles* que conforman el modelo *Aalaadin* (Fig.3.1). Una organización en este modelo es vista como un marco de actividades e interacciones a través de la definición de grupos, roles y sus relaciones. Así, una organización se puede describir únicamente en base a su estructura,

es decir, por la manera en la que se conforman sus grupos y roles para formar un todo, sin preocuparse por el comportamiento actual de los agentes [Gutknecht y Ferber, 2000].

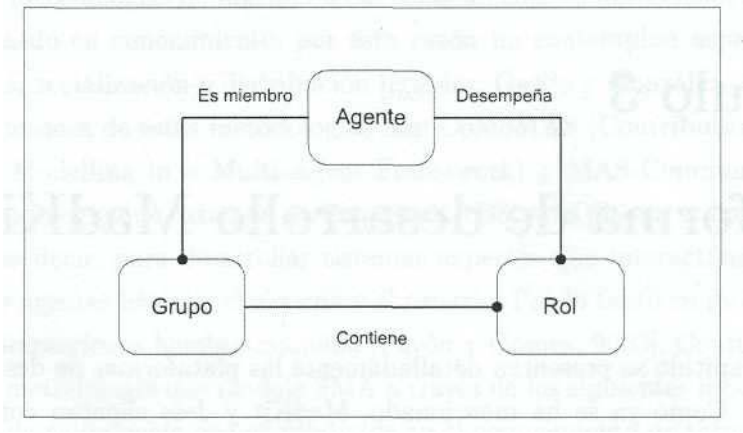

Figura 3.1: *Modelo Conceptual Aaladin de MadKit (Modelo reproducido de [Gutknecht y Ferber, 2000])*

A continuación se explican los elementos del modelo *Aalaadin:*

■ ***Agente***

El modelo *Aalaadin* no pone ninguna restricción con respecto a las arquitecturas internas y no asume ningún formalismo para agentes individuales. Un *agente* es especificado solamente como una entidad comunicativa activa que tiene *roles* dentro de los *grupos* existentes.

■ ***Grupo***

Los *grupos* son definidos como conjuntos atómicos de agregación de *agentes*. Cada *agente* puede ser parte de uno o más *grupos*. En su forma más básica, el *grupo* es sólo una forma de etiquetar a un conjunto de *agentes,* y en una forma más desarrollada, en conjunción con la definición del *rol,* el *grupo* permite representar cualquier Sistema Multiagente.

Los *grupos* tienen las siguientes características:

- Un *agente* puede ser miembro de diversos *grupos* al mismo tiempo.
- Los *grupos* pueden traslaparse libremente.

- Un *grupo* puede ser fundado por cualquier *agente,* y un *agente* puede solicitar la admisión a cualquier *grupo,* independientemente de si puede ser aceptado o no.
- Un *grupo* puede ser local o distribuido en varias máquinas.

■ ***Rol***

El rol es una representación abstracta de la función, servicio o identificación de un agente dentro de un grupo. Cada agente puede manejar uno o más roles, y cada rol puede ser manejado por un agente local o un grupo.

También se considera que, a nivel de diseño de un sistema multi-agente, las interacciones entre los agentes pueden ser abstraídas definiendo la interacción entre los escenarios y los roles. Este modelo permite a los agentes manejar varias situaciones de diálogo heterogéneas simultáneamente.

Un rol tiene las siguientes características:

- *Singularidad:* Un rol puede ser único o múltiple dentro de un grupo. Un rol identificado como único debe ser tomado por un sólo agente dentro de un grupo dado.
- *Competencia:* Una competencia identifica la condición que un agente debe satisfacer para ser capaz de jugar un rol dentro de un grupo.
- *Capacidad:* Una capacidad es la propiedad que se le da a un agente cuando juega un rol.

Estas propiedades no corresponden necesariamente a una implementación específica dentro de una plataforma, pero podrían ser adoptadas como una guía metodológica cuando se definen grupos y roles dentro de un sistema.

Un rol especial en un grupo es el rol de *administrador de grupo,* que es otorgado automáticamente al agente creador del grupo; éste tiene la responsabilidad de manejar las solicitudes de admisión al grupo o las solicitudes de rol y además puede revocar tanto roles como miembros del grupo.

En este modelo hay cuatro posibles mecanismos para acceso a un rol o grupo:

■ *Decisión automática:* Puede ser aceptado o rechazado de forma sistemática.

■ *Restringido por implementación:* Cuando el candidato debe exhibir alguna interfaz para ser autorizado a unirse al grupo.

- *Condicionado a un diálogo de admisión:* Cuando la solicitud induce a una interacción entre el administrador y el candidato para negociar la admisión.
- *Por una métrica a otro agente:* Definiendo un coeficiente de similaridad entre los miembros del grupo actual y el candidato.

3.1.2. Arquitectura de la Plataforma MadKit

Además de los conceptos principales (agente, grupo y rol), la plataforma especifica tres principios de diseño que se representan en la Arquitectura de la Plataforma MadKit (Fig.3.2): [Ferber y Gutknecht, 1998]

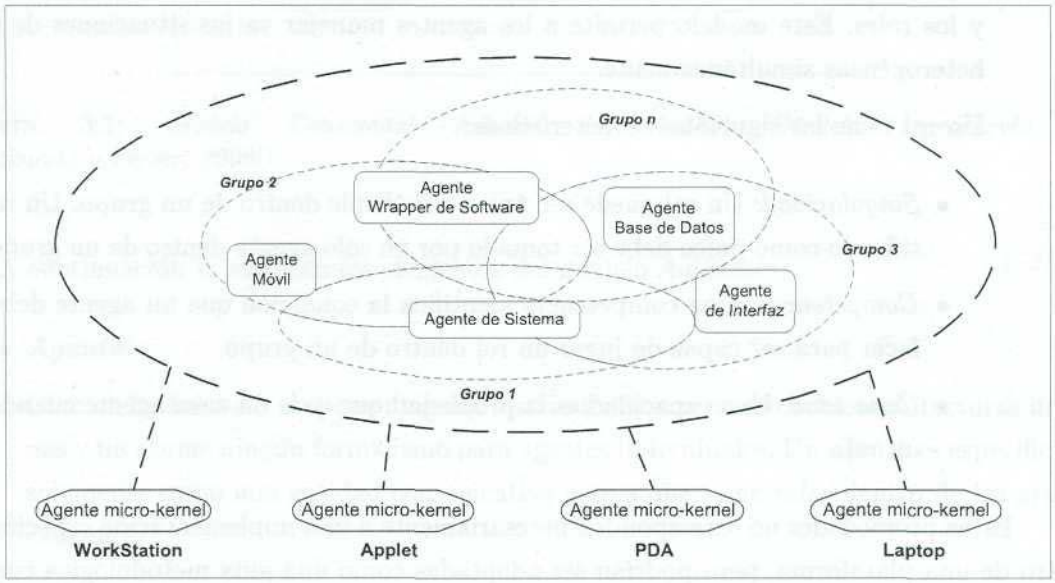

Figura 3.2: *Arquitectura de la Plataforma MadKit*

- *Arquitectura micro-kernel.* El término "micro-kernel" es utilizado como una referencia al papel del micro-kernel en el dominio de la ingeniería de Sistemas Operativos. *El microkernel incorpora un número de recursos clave que permitan un despliegue eficiente de los agentes.*

  Las tareas del Micro-Kernel son:

  • Control de los roles y grupos locales.

- Admisión del ciclo de vida del agente.
- Paso local de mensajes.

Agentes, grupos y roles en la plataforma:

Los agentes son definidos por herencia desde una **clase abstracta Agent**, que proporciona la identificación del agente, el API de los mensajes y llamadas relacionadas a grupos y roles. Estos métodos ofrecen la creación de grupos, unión y varias llamadas para encontrar qué roles se presentan en un grupo, qué agentes tienen un rol determinado, solicitar el manejo de un rol, la delegación o su retiro. Cada agente se ejecuta sobre su propio thread de control.

Paso de mensajes:

La clase de información estándar que se intercambia en MadKit es de tipo **Mensaje**, a través del cual se definen elementos como: emisor, receptor o fecha de emisión. Los mensajes receptores y emisores son identificados mediante la clase AgentAddress.

MadKit proporciona diversos tipos de mensajes predefinidos como: ActMessage, KernelMessage, ObjectMessage, ReactiveAgentMessage, StringMessage y XMLMessage. Así, los mensajes específicos pueden ser definidos por comunicación intra-grupos, y permiten al grupo tener sus atributos específicos de comunicación.

- ***Agentificación de servicios.*** La arquitectura de MadKit utiliza los agentes para lograr el paso de mensajes distribuidos, control de migración, seguridad dinámica y otros aspectos de administración de sistemas. Estos diferentes servicios son represen tados en la plataforma como roles "únicos" en algunos grupos específicos. Esto permite un alto nivel de personalización, y esos servicios de agentes pueden ser reemplazados sin ningún problema.

- ***Modelo del componente gráfico.*** El modelo gráfico de ésta plataforma está basado en componentes gráficos independientes, es decir, cada agente es responsable solamente de su propia interfaz gráfica.

3.2. Jess: Java Expert System Shell

Jess, es un shell para el desarrollo de SE y un lenguaje script desarrollado completamente en lenguaje Java, por Ernest Friedman-Hill en Sandia National Laboratories en

Livermore, CA. Jess fue inspirado originalmente por el Shell de Sistemas Expertos CLIPS [Friedman, 2000].

Jess es una herramienta para la construcción de *Sistemas Expertos.* Además, puede crear applets Java y aplicaciones que tengan la capacidad de *razonar* utilizando conocimiento proporcionado por el usuario, en forma de reglas declarativas. Este shell utiliza el *algoritmo RETE* para el procesamiento de las reglas, así como también proporciona su propia notación para la definición de reglas, su sintaxis incluye [Friedman, 2000]:

- *Átomos:* También llamados símbolos, son muy parecidos a los identificadores en otros lenguajes. Un átomo Jess puede contener letras, números y los siguientes símbolos: $, *, —, +, /, < , > , _ , ? , # .

- *Números:* Jess analiza sólo números de punto flotante y enteros. No acepta notación científica o de ingeniería.

- *Cadenas:* Las cadenas de caracteres en Jess son denotadas por comillas (").

- *Listas:* Consiste en un conjunto encerrado en paréntesis y cero o más átomos, números, cadenas u otras listas.

- *Comentarios:* Los comentarios en Jess se denotan por medio del punto y coma (;), haciendo respetar hasta el final de la línea todo el texto que le siga.

- *Funciones:* Las llamadas de funciones en Jess son listas simples y utilizan notación prefija, por ejemplo, si una expresión utiliza la función + para adicionar dos números, puede ser escrita como: (+ 2 3).

- *Variables:* Las variables en Jess son átomos que comienzan con el carácter del signo de interrogación, el cual es parte del nombre de la variable como: ?x. También se pueden definir multivariables, que puede referirse a un tipo especial de lista llamada *multicampo* y se define mediante el signo: $, ejemplo: $?X.. Para asignar cualquier variable, se utiliza la función *bina:*

  Ejemplos:

  (bind ?a 123), quiere decir que "a" tendrá el valor: (123).

  (bind $?lista-abarrotes (créate $ leche pan carne)), dará como resultado una lista, ya que se definió como multivariable: (leche pan carne).

Una *base de conocimientos* es una colección de partes de conocimiento llamadas hechos (facts). En Jess existen varios tipos de *facts:*

Ordered facts (Hechos ordenados): son listas, donde el primer campo (la cabeza de la lista) actúa como un tipo de categoría para el hecho. Los hechos ordenados se pueden adicionar a la base de conocimientos utilizando la función *assert.*

Ejemplo: (assert (Asesor Lety Ramón))

A cada hecho se le asigna un índice entero, que es el identificador del hecho (fact-id), cuando es afirmado.

Unordered facts (Hechos desordenados): Los hechos ordenados son útiles, pero no son estructurados. En los lenguajes orientados a objetos, los *objetos* tienen atributos que al ser inicializados representan datos concretos. Los hechos desordenados ofrecen esta capacidad mediante los llamados *slots,* que juegan el papel de atributos.

Antes de crear los hechos desordenados correspondientes, se deben definir los *slots* que se van a utilizar mediante el constructor *deftemplate.* Puede haber un número arbitrario de *slots* y cada uno de ellos debe ser un átomo, éstos son como los identificadores en otros lenguajes.

Ejemplo:

(deftemplate automóvil "Un auto específico."
(slot hecho-en)
(slot modelo)
(slot año (tipo INTEGER))
(slot color (default blanco))

Constructor deffacts: Este constructor permite definir una lista de hechos, en lugar de definirlos por separado cada uno, esta lista es adicionada dentro de la base de conocimientos.

Hechos Definstance: A diferencia de defclass, que genera un deftemplate especial para representar una categoría de Beans, definstance sólo representa un Bean específico sobre la base de conocimientos. Un Java Bean es un componente de software reutilizable como: botones, campos de texto, cajas de listas, barras de desplazamiento o diálogos; y pueden ser manipulados visualmente mediante herramientas de construcción como por ejemplo: Visual Basic, Visual Age o Delphi.

La representación de los beans puede ser *estática* (no cambia frecuentemente, como lo haría una propiedad en un punto en el tiempo) o *dinámica* (cambia de forma automática siempre que las propiedades del Bean cambien).

Antes de insertar cualquier Bean en la base de conocimientos, se necesita un deftemplate para representarlo, por lo que se necesita utilizar defclass para decirle a Jess que lo genere.

Después de lo anterior, se puede usar la función *definstance* para adicionar el objeto en la base de conocimientos.

Jess lleva a cabo la *adquisición de conocimiento* mediante su consola por parte de constructores como *deftemplate* y *deffacts* (Figura 3.3).

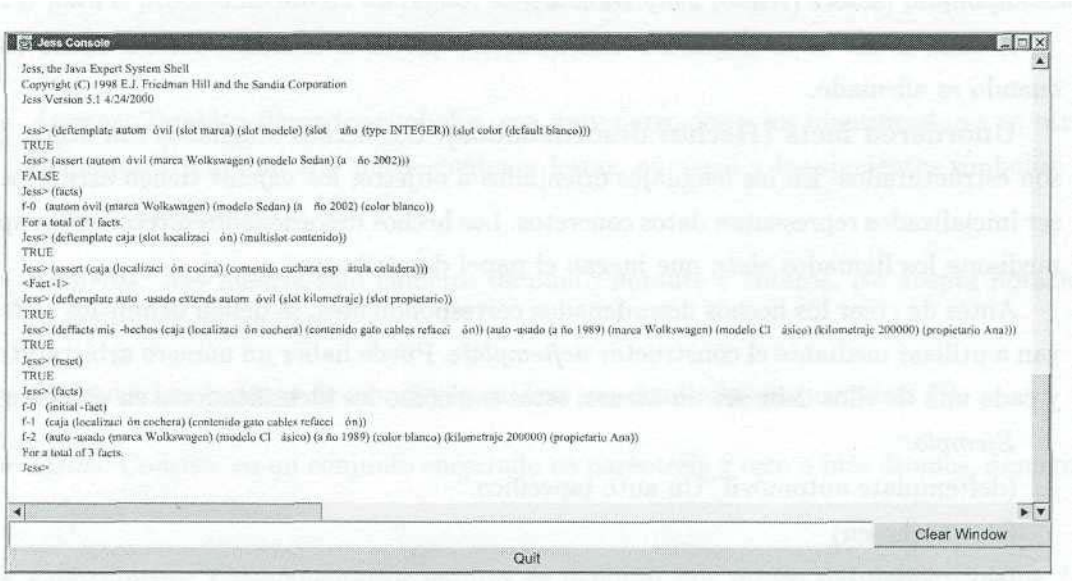

Figura 3.3: *Ejemplo de la construcción de una Base de Conocimientos con sus hechos, mediante deftemplate y deffacts.*

Para que Jess haga esto de una forma *automática* y se tenga la posibilidad de salvar conocimiento dentro de una base de datos, se emplean archivos **.clp,* estos archivos contienen hechos y reglas que el usuario desea proporcionar a Jess como conocimiento para su sistema.

Existen varios algoritmos de búsqueda a través de las reglas para inferir conclusiones a partir de los hechos y las reglas. Todos los algoritmos son del tipo "pattern-matching" (asociación de patrones), van disparando reglas a medida que se cumplen las condiciones.

Ya que el objetivo principal de Jess es la ejecución de reglas mediante declaraciones de la forma if-then, Jess tiene que verificar constantemente si la declaración if es verdadera, entonces ejecutará su correspondiente declaración then. Para hacer esto de una forma correcta, la Máquina de Reglas de Jess utiliza el algoritmo RETE [Friedman, 2000].

Algoritmo RETE

Este algoritmo es un método de comparación de un conjunto de patrones contra un conjunto de objetos para determinar todas las posibles coincidencias. Originalmente, este algoritmo se desarrolló para ser utilizado en sistemas de producción. Un sistema de producción consiste en una colección de declaraciones if-then llamadas *producciones,* los datos operados sobre las producciones son retenidas en una base de datos global llamada *memoria de trabajo.*

En este algoritmo la declaración ¡f es llamada LHS (left-hand side) y la parte then llamada RHS (right-hand side). La memoria de trabajo contiene objetos con pares asociados de atributos-valores. El LHS de una producción consiste de una secuencia de patrones, esto es, una secuencia de descripciones parciales de los elementos de la memoria de trabajo [Friedman, 2000].

Este algoritmo de comparación de patrones está diseñado para aprovechar la redundancia temporal, lo que hace al guardar el estado del proceso de comparación de un ciclo a otro y luego calculando de nuevo los cambios en este estado sólo para los cambios que suceden en la lista de hechos. Es decir, si un conjunto de patrones encuentra dos de los tres hechos necesarios en un ciclo, no se requiere de una verificación durante el siguiente ciclo para los dos hechos que ya se encontraron, sólo el tercer hecho es de interés. Al tipo de información de estado que indica los hechos que tienen patrones coincidentes en una regla se le llama *coincidencia parcial.* Al otro tipo de información de estado guardada se le llama **coincidencia de patrón,** que se presenta cuando un hecho ha satisfecho un patrón individual de cualquier regla sin considerar las variables de otros patrones que puedan restringir el proceso de coincidencia [Giarratano y Riley, 2001].

La comparación de hechos se divide en dos pasos:

1. Cuando se agregan y se eliminan hechos debe determinarse cuáles patrones han coincidido, esto se realiza en la *red patrón.*

2. Debe verificarse la comparación de las uniones de variables mediante patrones para determinar las coincidencias parciales de un grupo de patrones, esto se efectúa en la ***red de unión.***

La red patrón se organiza de forma jerárquica, colocando en la parte superior los nodos patrón que corresponden a las primeras restricciones de ranura de los patrones. Cuando se afirma un hecho, se revisan los nodos patrón para las primeras restricciones de ranura de

la red patrón. Cualquiera de ellos cuya especificación de comparación se satisface, activará los nodos patrón que se encuentran directamente debajo de él. Este proceso continúa hasta alcanzar un nodo terminal de la red, que representan el final de un patrón y una coincidencia de patrón exitosa. Cada nodo terminal tiene una memoria **alfa** o **derecha** asociada a él, que contiene el conjunto de todos los hechos que han coincidido con el patrón asociado al nodo terminal [Giarratano y Riley, 2001].

En la Fig.3.4, se muestran dos reglas que se representan el la red patrón de la Fig.3.5.

```
(defrule regla-1-Rete
        (comparar (un rojo))
        (datos (x ?x) (y ?x))
        =>)
(defrule regla-2-Rete
        (comparar (a ?x) (b rojo))
        (datos (x -verde) (y ?x))
        (datos (x ?x) (y ?x))
        =>)
```

Figura 3.4: *Reglas para comparar.*

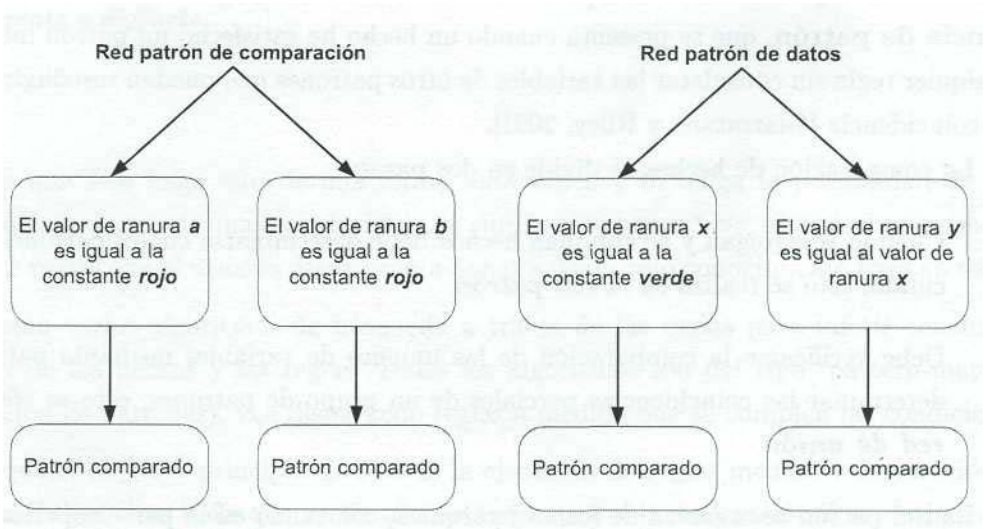

Figura 3.5: *Red patrón para dos reglas.*

Una vez que se han determinado los patrones que han coincidido con los hechos, debe verificarse la comparación de los enlaces de variables a través de los patrones para cerciorarse que las variables usadas en más de un patrón tienen valores congruentes. Esta comparación se efectúa en la ***red de unión,*** cada nodo terminal de la red patrón actúa como una entrada a una **unión** o **nodo de dos entradas,** de la red de unión. Cada unión contiene una especificación de comparación para las coincidencias de la memoria alfa asociada con su nodo terminal y para el conjunto de coincidencias parciales que han comparado patrones anteriores. Esas coincidencias parciales se almacenan en la memoria **beta** o **izquierda** de la unión.

La primera unión compara los primeros dos patrones y el resto de las uniones compara un patrón adicional con las coincidencias parciales de la unión previa; por ejemplo, dada la Regla-2-Rete (Fig.3.6), utilizada en el ejemplo anterior:

```
(defrule regla-2-Rete
        (comparar (a ?x) (b rojo))
        (datos (x -verde) (y ?x))
        (datos (x ?x) (y ?x))
        =>)
```

Figura 3.6: *Ejemplo Regla-2-Rete.*

La primera unión contendría la especificación de comparación:

*El valor de ranura **a** del hecho limitado al primer patrón es igual a el valor de ranura y del hecho limitado al segundo patrón.*

La segunda recibiría como entrada el conjunto de coincidencias parciales de la primera unión y contendría la siguiente especificación de comparación:

El valor de ranura x del hecho limitado al tercer patrón es igual a el valor de ranura y del hecho limnitado al segundo patrón.

Se observa que la variable ?x del tercer patrón podría tener su valor comparado con la variable ?x del primer patrón, en lugar de compararse con la variabla ?x del segundo patrón. La segunda aparición de ?x en el tercer patrón no tiene que verificarse en la red de unión porque es posible hacerlo en la red de patrón. En la Fig.3.7, se muestran las redes patrón y de unión para la regla Regla-2-Rete del ejemplo.

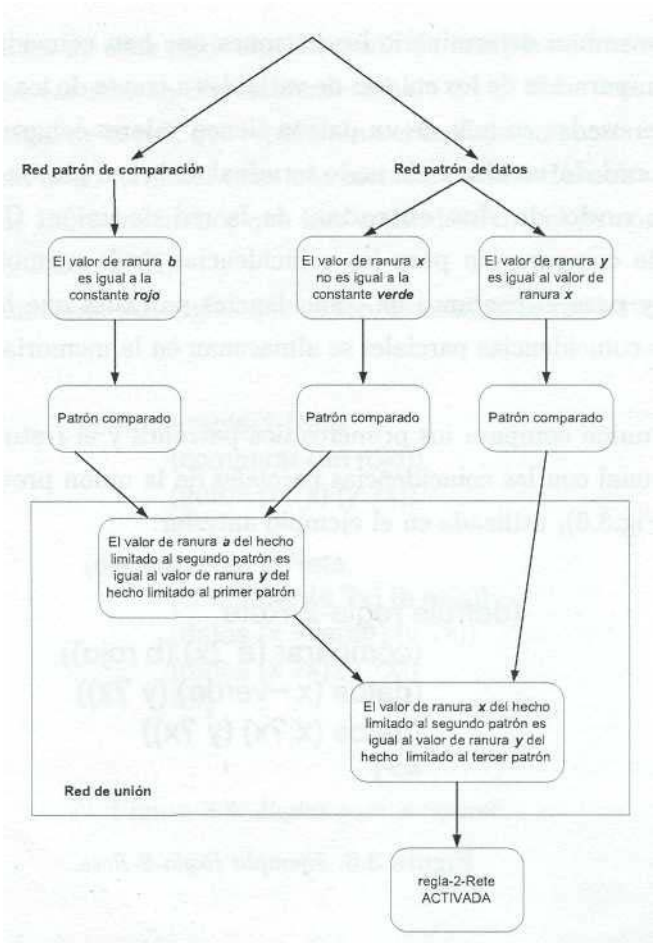

Figura 3.7: *Redes patrón y unión para la regla-2-Rete.*

Para el algoritmo Rete, las reglas se convierten en estructuras de datos en una red de reglas, que consta de una red patrón y una red de unión; la primera compara hechos con patrones y la segunda verifica que las uniones de variables a través de los patrones sean congruentes [Giarratano y Riley, 2001].

3.2.1. Especificaciones de Jess

Jess proporciona una API (Application Programming Interface), que facilita su utilización en la fase de programación. Sin embargo, la documentación disponible para la fase de análisis y diseño es deficiente. Por ello, como parte de este trabajo se desarrollaron el Diagrama de

Clases de Jess (Fig.B.l), y un cuadro descriptivo de las relaciones entre las clases (Apéndice B.3).

En el diagrama de clases de Jess que se elaboró, no se especifican los métodos y atributos de las clases debido a la magnitud de los mismos, pero sí están representadas todas las relaciones y generalizaciones entre ellas.

3.3. MadKit-Jess

En las secciones anteriores se detallaron las plataformas a utilizar para este trabajo. Se requiere la integración de ambas plataformas para lograr el objetivo de este trabajo. A través de las investigaciones realizadas, se encontró un proyecto llamado JessAgentLib. Éste es un conjunto de librerías que tiene como objetivo la ejecución de Jess dentro de MadKit (Fig.3.8).

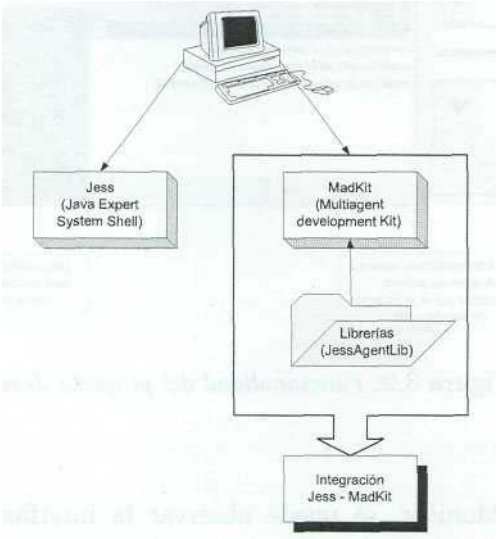

Figura 3.8: *Integración del proyecto JessAgentLib*

Se encontró que este proyecto funciona con tres tipos de agentes: JessAgent, JessMonitor y EditJessAgent. Los objetivos de estos agentes son: activar un agente que permite al usuario visualizar la descripción de la integración (JessAgent), un editor que permite ejecutar comandos Jess a través de su interfaz gráfica (EditJessAgent) y un monitor que controla las direcciones de los agentes, mostrando qué agentes son los que están activos (JessMonitor) (Fig.3.9).

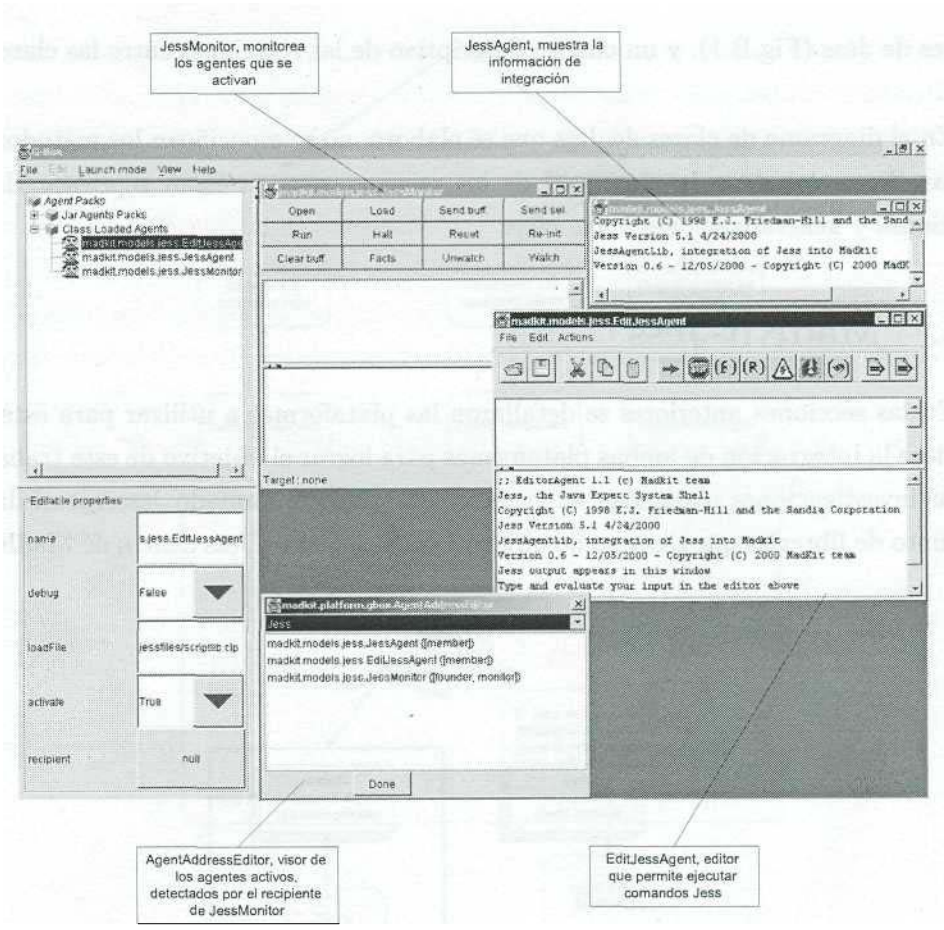

Figura 3.9: *Funcionalidad del proyecto JessAgentLib*

En el agente JessMonitor, se puede observar la interfaz gráfica con un conjunto de botones, de los que únicamente funciona la opción *Open.* La documentación disponible sobre el funcionamiento de este agente, indica que es el encargado de monitorear cualquier agente JessAgent que se active, registrando su dirección (a través del método AgentAddress) en un contenedor, para controlar a estos agentes (mediante el método ControlMessages). Sin embargo, al ejecutar estos agentes para su análisis, se encontró que las direcciones de los agentes JessAgent y EditJessAgent se registran correctamente en el contenedor sin ejecutar al agente JessMonitor, por lo que no dependen de él para su funcionamiento.

También se determinó, que no existe una comunicación entre estos agentes a través del intercambio de mensajes. El agente EditJessAgent, cuenta con una interfaz gráfica por medio

de la cual se ejecutarán comandos Jess, pero no requiere que alguno de los otros agentes estén activos para su funcionamiento. Por lo tanto, no se detectó diferencia funcional entre ejecutar el agente Edit JessAgent en el ambiente MadKit y ejecutar la interfaz de consola que proporciona el shell Jess, ambos ofrecen las mismas opciones.

El análisis que se realizó con este proyecto, ayudó a conocer sus limitaciones, así como también determinar que no se podía partir de esos agentes, debido a su nula interacción. También, debido a la complejidad del código de Jess (19935 líneas) resultaba más difícil corregir los errores y limitaciones en JessAgentLib, que desarrollar el proyecto partiendo de las plataformas separadas. Así, se decidió analizar de forma independiente la máquina de inferencias utilizada por Jess (RETE) y algunas de sus utilerías, para lograr la interacción adecuada entre los agentes expertos.

La interacción entre los agentes expertos que se propone en este trabajo, se logró con el desarrollo del modelo de cooperación que se presenta en el siguiente capítulo.

Capítulo 4

Análisis y Diseño del Sistema Cooperativo de Agentes Expertos (SiCAE)

En el presente capítulo se describen las fases de análisis y diseño del Sistema Cooperativo de Agentes Expertos (SiCAE). Así mismo, se define el Modelo de Cooperación en Sistemas Multiagentes (SMA) utilizado en este trabajo

4.1. Modelo de cooperación

La estrategia utilizada en el presente trabajo, consiste en descomponer y distribuir tareas complejas, logrando tener subtareas más pequeñas que requieran menos complejidad en el agente [Weiss, 1999].

La descomposición de las tareas puede hacerse de forma *espacial,* basada en el esquema de las fuentes de información o de los puntos de decisión, o de forma *funcional* de acuerdo al área de especialización de los agentes [Weiss, 1999].

Una vez que las tareas se han descompuesto, son distribuidas de acuerdo a los siguientes criterios:

- Evitar una carga excesiva de los recursos.
- Asignar tareas a los agentes con igualdad de capacidades.
- Hacer que un agente con un rol determinado, asigne las tareas a los demás agentes.
- Asignar tareas independientes a los agentes en una proximidad espacial o semántica. Esto minimiza los costos de comunicación y sincronización.

■ Reasignar tareas, si es necesario, para completar tareas urgentes.

El modelo de cooperación propuesto, utiliza la distribución de tareas de forma funcional y el mecanismo de estructura organizacional (Fig.4.1).

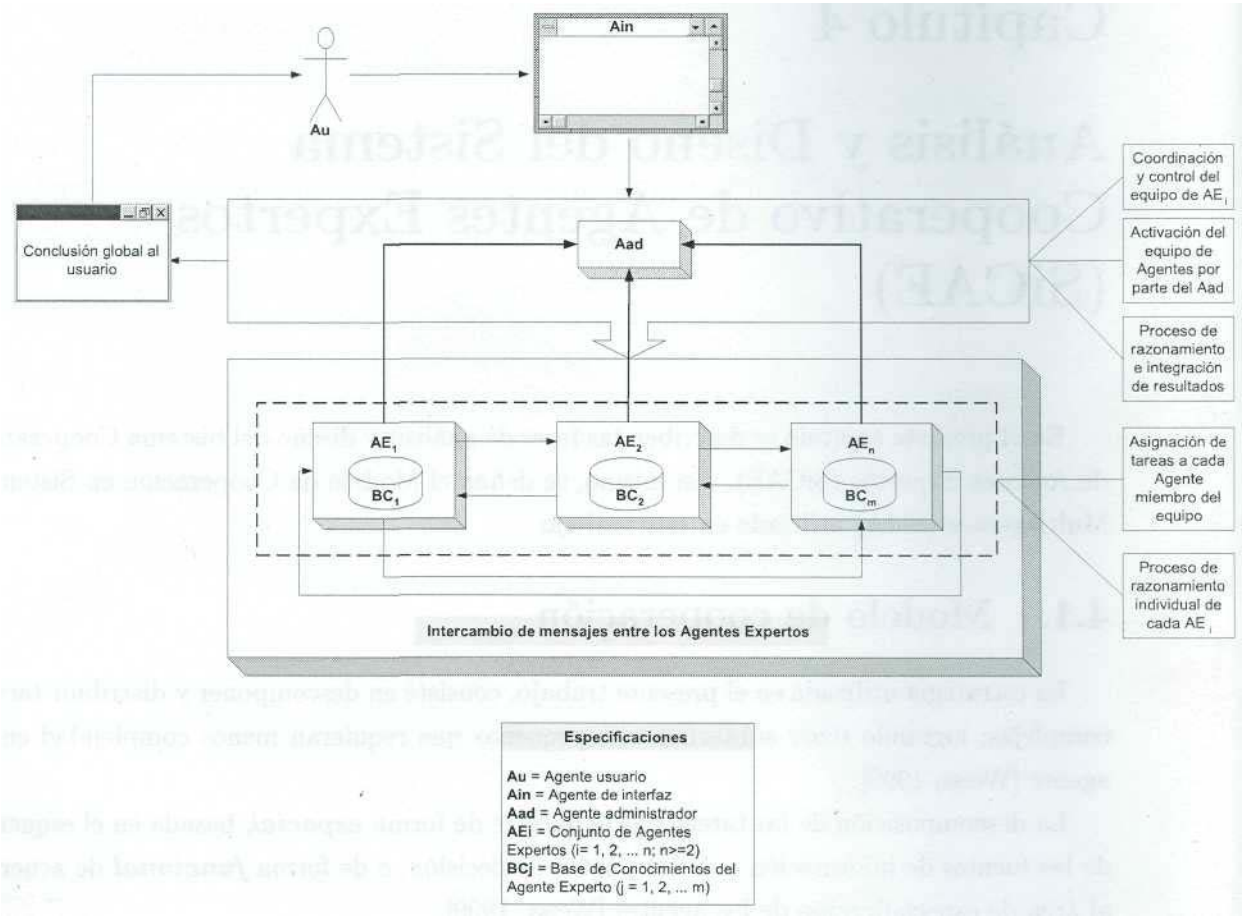

Figura 4.1: *Diagrama del modelo de cooperación propuesto.*

Para representar el modelo, se requiere identificar los siguientes puntos:

Problema:

Se plantea el caso de un equipo de especialistas en algún dominio débilmente estructurado. Se tomará como caso de estudio un equipo de expertos médicos para diagnosticar enfermedades, el cual se detalla en el siguiente capítulo.

Subproblemas:

Esta es la fase de descomposición de tareas, que en el presente trabajo consiste en descomponer el conocimiento del caso de estudio en módulos. Se deben identificar las actividades a realizar por cada elemento que conforme el equipo, de acuerdo a su rol asignado.

Los tipos de agentes existentes en el modelo son: Agente usuario, Agentes expertos, Agente de interfaz y Agente administrador.

Asignación de Tareas:

Una vez identificados los subproblemas y elementos, se procederá a aginar las tareas correspondientes a cada elemento.

- **AE_i:** Es el conjunto de agentes que tienen asignado el conocimiento correspondiente a su área de experiencia. Cada agente experto activo en el framework tiene asignada una **BC_j,** que será procesada internamente por la máquina de inferencias.

- **Ain:** El Agente de interfaz tiene la tarea de mantener la comunicación entre el usuario y el resto de los elementos del sistema.

- **Aad:** El Agente administrador tiene las siguientes tareas: coordinación y control, activación del equipo de **AE_i** e integración de resultados.

El diagrama del modelo de cooperación propuesto, se describe como sigue:

Una vez que el Au ha inicializado al Ain, se tienen tres partes principales: la primera es la **Coordinación y control del equipo de AE_i,** llevada a cabo por el Aad, responsable de la activación del equipo de Agentes. En el momento de la activación entra la segunda parte, **la Asignación de tareas a cada Agente miembro del equipo,** consiste en activar a cada AE_i con su correspondiente BC_i, que representa el área de conocimiento y experiencia de cada uno, en esta etapa se llevan a cabo los intercambios de mensajes entre los AE_i activos, que representa la cooperación entre ellos. Cada uno de los AE_i obtendrá su conclusión individual en base al proceso de cooperación y deberán enviar ese resultado al Aad para que se lleve a cabo la tercera parte y la mas compleja, **Proceso de razonamiento e integración de resultados.** El Aad integrará los resultados mediante un mecanismo de razonamiento que permita obtener una conclusión global.

4.2. Mecanismo de comunicación y cooperación

El mecanismo de comunicación y cooperación entre los AE_i del modelo propuesto, está dado principalmente por la integración de las dos plataformas de desarrollo establecidas en este trabajo, MadKit y Jess.

La forma en la que interactúan los AE_i está definida por la plataforma MadKit, a través de su arquitectura organizacional Aalaadin y el proceso de razonamiento se lleva a cabo mediante la máquina de inferencia de Jess. Jess utiliza el algoritmo RETE para realizar el mecanismo de inferencia, el cual emplea el encadenamiento hacia adelante (forward chaining) para un eficiente emparejamiento de patrones.

El algoritmo RETE crea un árbol de decisión que combina los patrones en todas las reglas de la base de conocimiento. Una vez que ha sido determinado qué patrón ha sido emparejado por los hechos, las comparaciones de las variables asignadas a través de los patrones deben ser revisadas en una memoria de trabajo que "recuerda" qué emparejamientos parciales ya han sido probados. La eficiencia de este algoritmo está en hacer un emparejamiento parcial, que recuerda los resultados de pruebas pasadas a través de las iteraciones del ciclo de emparejamiento de las reglas, es decir, sólo prueba nuevos hechos.

Cada nodo en un grafo o red RETE representa el conjunto de variables asignadas que emparejan una afirmación o una colección de afirmaciones. Una ruta a través del grafo hacia una hoja, representa las asignaciones que emparejan los antecedentes a una regla. La construcción de un grafo RETE se lleva a cabo de la siguiente manera:

- Para cada antecedente, se-crea un nodo "alfa", que es un nodo "emparejar" (son los nodos raíz).
- Se une un primer antecedente y un segundo antecedente para crear un nodo "beta", que es un nodo "unión".
- Se unen todos los antecedentes siguientes con el nodo unión previo, para crear un nuevo nodo unión.
- Para cada consecuente, se crea un nodo "terminal", con una acción final de ejecución para obtener un subconjunto de todas las variables emparejadas necesarias para llegar al consecuente.

Cada agente que es lanzado por el Aad, lo hace mediante un método propio de MadKit llamado launchAgent. Cuando un agente es lanzado ya tiene incorporado el conocimiento que

utilizará en su proceso de razonamiento, descrito en los párrafos anteriores. Esta etapa en el modelo es la de **Asignación de tareas al equipo de agentes expertos.** Una vez asignadas las tareas, los AE_i llevan a cabo un proceso de cooperación, intercambiando mensajes que contienen el conocimiento procesado por cada uno de ellos. MadKit proporciona sus propios métodos para intercambio de mensajes entre agentes, estos métodos están regidos por una clase principal llamada AbstractAgent, que es la encargada de controlar el ciclo de vida de los agentes, mensajes, interfaz gráfica, administración de grupos y roles e información de los agentes.

Este proceso de **Intercambio de mensajes entre los agentes expertos,** no sólo está controlado por la plataforma MadKit, sino que se desarrolló un mecanismo de integración con la máquina de inferencia para que la información intercambiada corresponda al conocimiento que cada AE_i tiene asignado. Cuando este proceso ha terminado, se lleva a cabo el **Proceso de razonamiento individual de cada** AE_i, que permite que cada agente obtenga su conclusión mostrándola al Au.

4.3. Análisis del SiCAE

En esta fase de análisis, se abordan los puntos siguientes:

1. **Análisis del dominio:** Aquí se debe hacer una descripción del escenario general e identificación de dominios.
2. **Diagrama de casos de uso:** Se utiliza para representar las interacciones del sistema con su entorno, en este caso los actores son los diferentes roles identificados y los casos de uso, son la funcionalidad que ofrece el sistema.

4.3.1. Análisis del dominio

En esta parte del análisis se determina el Escenario General, que es donde se representa de forma general la interacción de los elementos, donde se identifican los dominios. Posteriormente se identifican los actores y roles, determinando sus funciones, capacidades y comunicación.

Escenario General

El diálogo entre el usuario y los agentes expertos cooperativos de esta aplicación es el siguiente:

1. Un usuario tiene la necesidad de hacer una consulta a un equipo de expertos para la solución de un problema o lograr un objetivo.
2. El usuario inicializa al Agente de interfaz, que es el encargado de inicializar al Agente administrador.
3. El Agente administrador inicializa a los agentes expertos para conformar el equipo y realizar la consulta.
4. En el momento en que cada agente experto es inicializado, es cargada la base de conocimientos correspondiente al dominio del conocimiento requerido para la consulta.
5. Cada agente experto muestra al usuario las preguntas correspondientes a cada base de conocimientos, a las mismas que el usuario podrá dar respuesta.
6. Los agentes expertos tienen la posibilidad de cooperar entre ellos dentro del equipo, para ayudar al usuario a tener una solución al problema proveniente de un consenso entre ellos.

Se hace la representación de este escenario general en la Fig. 4.2:

Identificación de dominios

En la Fig.4.2 se puede observar la existencia de dos dominios:

- *Dominio del usuario,* que es el que se encarga de hacer una consulta e inicializar al Agente de interfaz para lograr su objetivo.
- *Dominio de los agentes,* este dominio está conformado por un conjunto de agentes con roles diversos, es decir, tienen una función específica dentro del ambiente para integrarse en una pequeña sociedad donde estarán activos y llevarán a cabo sus objetivos.

De acuerdo a estos dominios, se identifican los actores y roles, definiendo sus funciones principales.

4.3.2. Identificación de actores y roles

Aquí se definen los actores y roles de acuerdo a los dominios identificados, estableciendo su función principal, comunicación y capacidades. Asimismo se muestra el Diagrama de Casos de Uso (Fig.4.3) para detallar las acciones de los actores.

El Diagrama de Casos de Uso se describe de la siguiente manera:

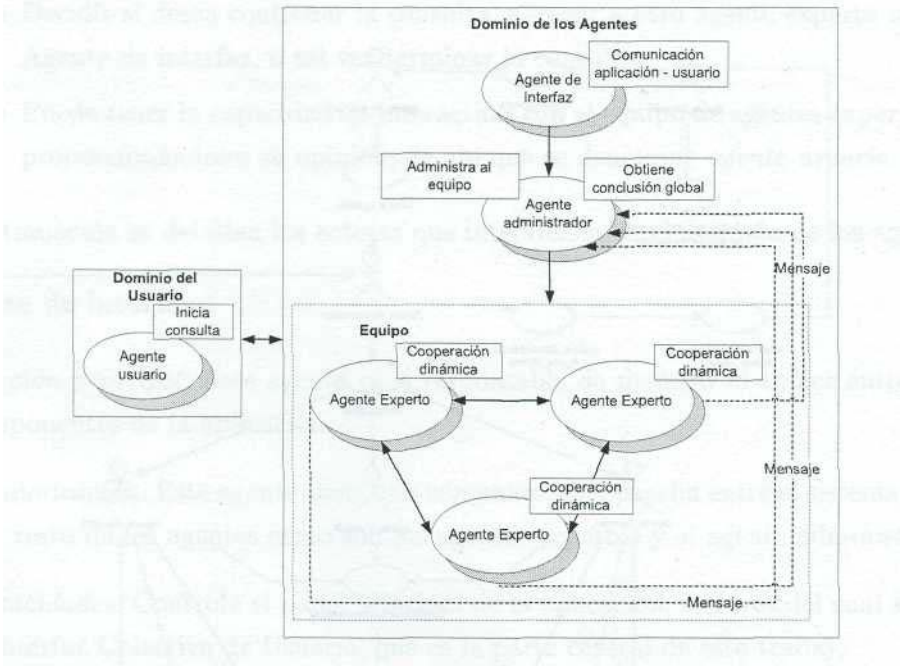

Figura 4.2: *Dominio General de la aplicación*

1. El Au inicia una consulta mediante el Ain.
2. A través del Ain se generarán a los agentes necesarios. El primero es el Aad, encargado de activar el equipo de AE_i, seleccionando la base de conocimientos deseada, esto lo hace tantas veces como agentes expertos desee en el grupo.
3. Cada Agente_Experto inicializado muestra al usuario la(s) pregunta(s) correspondientes a su base de conocimientos.
4. El Agente_Usuario responde a la(s) pregunta(s) mostradas por cada agente experto.
5. Los Agentes_Expertos intercambian mensajes de acuerdo a la respuesta recibida por parte del usuario. Al concluir esta parte de cooperación entre los Agentes_Expertos, se genera una conclusión que es mostrada al Agente_Usuario.

Agente usuario:

- *Función principal:* El usuario es el que inicia la consulta en la aplicación, y es el responsable de inicializar al Agente de interfaz.

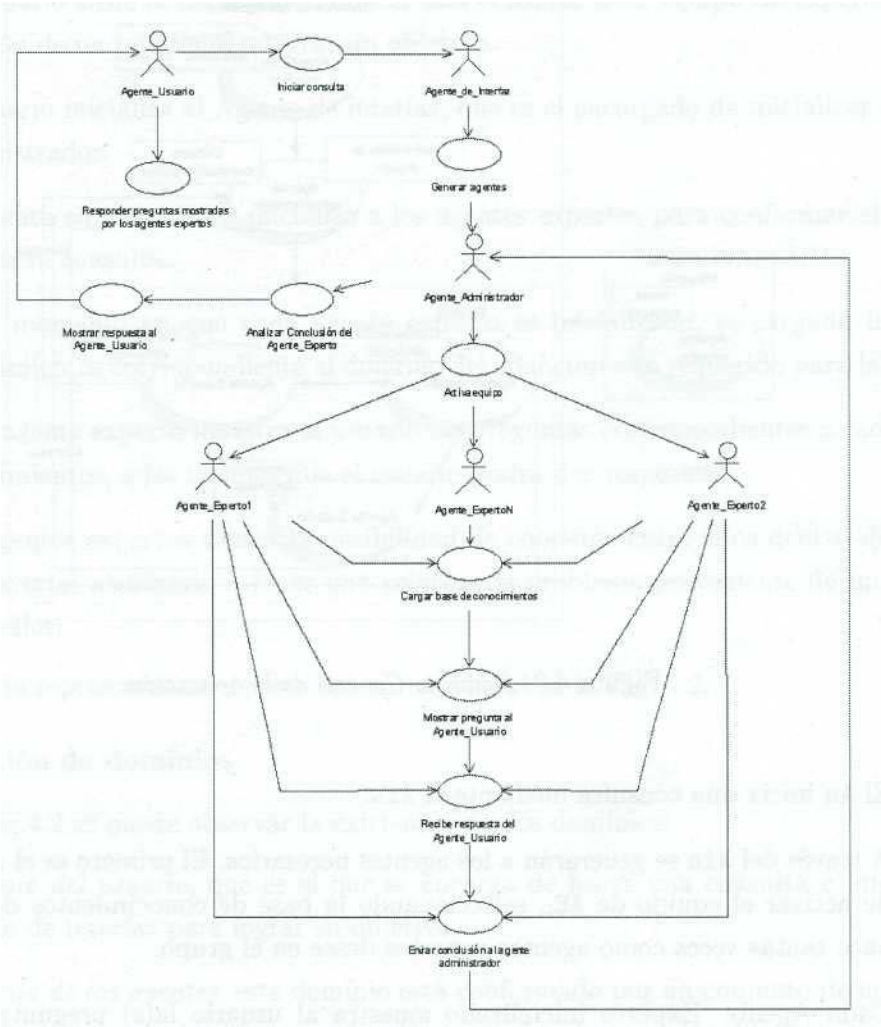

Figura 4.3: *Diagrama de casos de uso de la aplicación.*

- *Comunicación:* La comunicación que tiene el usuario, es por medio de la interfaz que proporciona el framework, a través del cual se inicializan los agentes expertos.
- *Capacidades:*
  - Iniciar la aplicación.
  - Inicializar al Agente de interfaz.

- Decidir si desea continuar la consulta, agregar a otro agente experto mediante el Agente de interfaz, o tal vez terminar la consulta.
- Puede tener la capacidad de interactuar con el equipo de agentes expertos creado, proporcionándoles su opinión, de ahí que se denomine *agente usuario.*

A continuación se detallan los actores que intervienen en el dominio de los agentes.

Agente de interfaz:

■ *Función principal:* Este agente es el responsable de manejar el enlace entre todos los componentes de la aplicación.

■ *Comunicación:* Este agente tiene una comunicación estrecha entre el sistema, el usuario y el resto de los agentes como son los agentes expertos y el agente administrador.

■ *Capacidades:* Controla el panel principal de la aplicación, a través del cual se inicializa la Interfaz Colectiva de Usuario, que es la parte central de este trabajo.

Agentes expertos:

■ *Función principal:* Estos agentes cargan la base de conocimientos indicada por el usuario y además de representar su dominio de experiencia, muestran al usuario las preguntas que éste deberá responder.

■ *Comunicación:* Tienen comunicación con Jess (Java Expert System Shell) para realizar la inferencia con las bases de conocimientos de cada uno de ellos y además, utilizan las clases definidas de comunicación de MadKit (Multi Agent Development Kit) para la comunicación entre ellos.

■ *Capacidades:*

- Cargar la base de conocimientos indicada por el usuario.
- Realizar proceso de inferencia de acuerdo a las respuestas dadas por el usuario.
- Comunicarse con los demás agentes expertos que integran el equipo, con la finalidad de que todos tengan conocimiento del siguiente nodo a procesar por el algoritmo RETE.

Agente administrador:

- *Función principal:* Activa el equipo de agentes expertos y mantiene la coordinación y control entre ellos.
- *Comunicación:* Este agente mantiene una comunicación con el agente usuario y los agentes expertos.
- *Capacidades:* Cuando el equipo está integrado, este agente tiene la tarea de recibir las conclusiones individuales de los agentes expertos que conforman el equipo, con el fin de realizar un proceso de razonamiento que permita obtener una conclusión global que será mostrada al usuario.

Este análisis es una parte imprescindible para la implementación de la aplicación, ya que de éste se desprenden todos los componentes que serán desarrollados. Todos los elementos establecidos en esta sección serán utilizados en el Diseño del Sistema Cooperativos de Agentes Expertos, que se detalla en la siguiente sección.

4.4. Diseño del SICAE

En esta sección se exponen los modelos diseñados, que ayudan a la implementación del framework. Los modelos son diseñados partiendo de los elementos establecidos en la fase de Análisis. Los modelos que se utilizan en esta fase son: Diagrama de Clases, Diagrama de Estados, Diagrama de Secuencia y Diagrama de Colaboración, cada uno se detalla en las secciones siguientes.

4.4.1. Diagrama de clases

Un diagrama de clases es una colección de elementos estáticos de un modelo, como son clases, interfaces y sus respectivas relaciones, conectados como un grafo entre sí y sus contenidos [Rendón, 2000].

El diagrama de clases del framework (Fig. 4.4) está basado en la Arquitectura Jess-MadKit y a continuación se explica.

Clase ExpertAgentMonitorPanel

Esta clase genera la interfaz gráfica con un panel principal. Este panel tiene cuatro opciones que el usuario puede utilizar: Inicializar experto, Abrir archivo, Limpiar área de

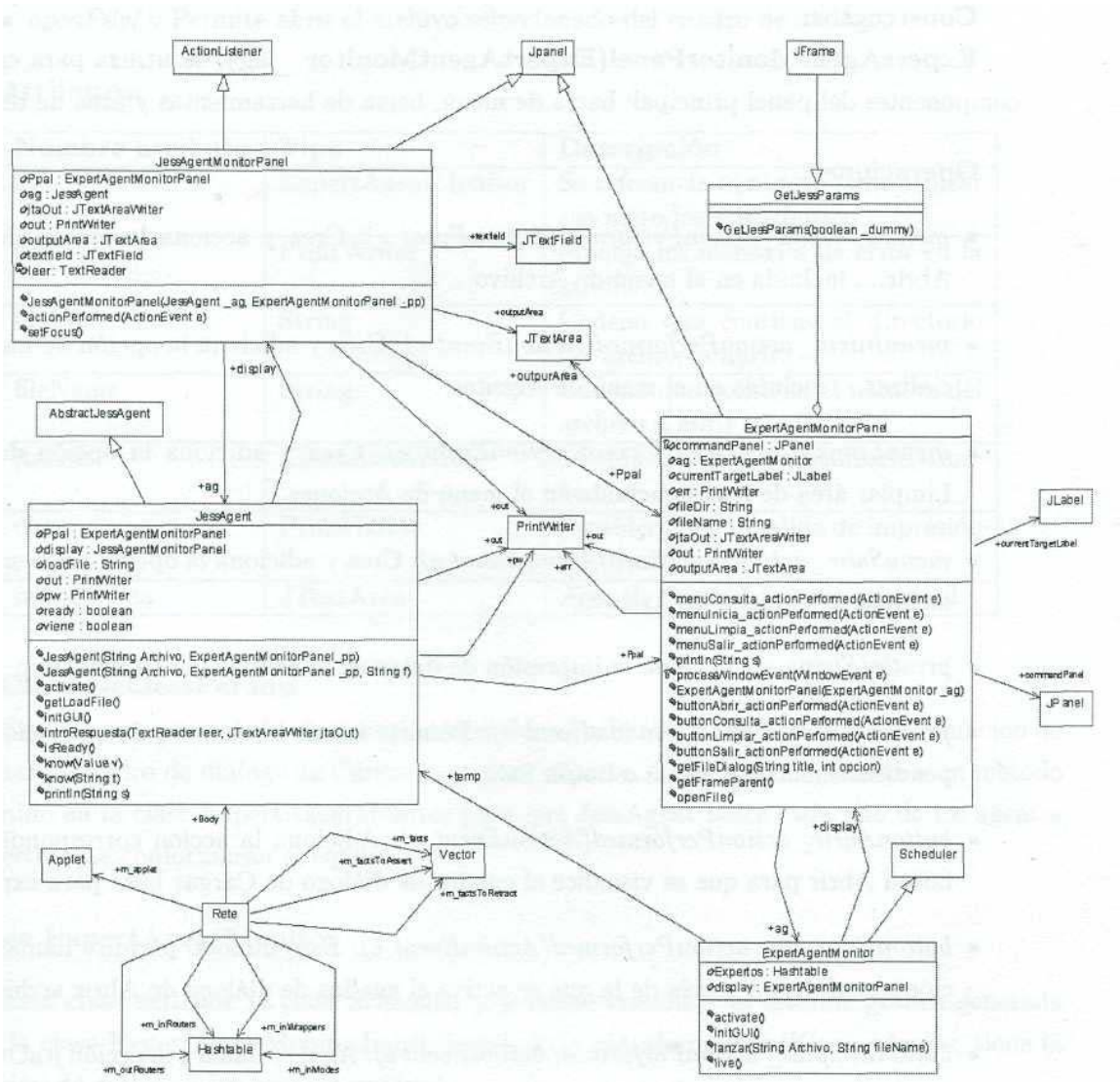

Figura 4.4: *Diagrama de Clases del framework desarrollado.*

texto y Salir, estas opciones están disponibles tanto en el menú de herramientas como en la barra de herramientas.

Constructor:
ExpertAgentMonitorPanel(ExpertAgentMonitor _ag), se utiliza para crear los componentes del panel principal: barra de menú, barra de herramientas y área de texto.

Operaciones:

- *menuConsulta_actionPerformed(ActionEvent e):* Crea y acciona la opción de menú Abrir..., incluida en el menú de Archivo.
- *menuInicia_actionPerformed(ActionEvent e):* Crea y adiciona la opción de menú Inicializar..., incluida en el menú de Agentes.
- *menuLimpia_actionPerformed(ActionEvent e):* Crea y adiciona la opción de menú Limpiar área de texto, incluida en el menú de Acciones.
- *menuSalir_actionPerformed(ActionEvent e):* Crea y adiciona la opción de menú Salir, incluida en el menú de Archivo.
- *println(String s):* Acciona la impresión de datos en la salida tipo PrintWriter.
- *processWindowEventfWindowEvent e):* Permite cerrar la ventana de aplicación inde pendientemente del menú o botón Salir.
- *buttonAbrir_actionPerformed(ActionEvent e):* Adiciona la acción correspondiente al botón Abrir para que se visualice el cuadro de diálogo de Cargar base para experto...
- *buttonConsulta_actionPerformed(ActionEvent e):* Este método permite llamar la ac ción openFileQ a través de la que se activa el cuadro de diálogo de Abrir archivo.
- *buttonLimpiar_ actionPerformed(ActionEvent e):* Aquí se llama a la acción jtaOut.clear() que indica que se limpiará el área de texto del panel principal.
- *buttonSalir_actionPerformed(ActionEvent e):* Método que adiciona al botón Salir la acción de salida del sistema de la aplicación.
- *getFileDialog(String tule, int opción):* Método que activa el cuadro de diálogo de Abrir archivo, ya sea mediante menú o botón.

- *getFrameParent():* Permite visualizar el cuadro de diálogo "Cargar base para exper tos..." y seleccionar el archivo *.clp requerido.

- *openFüef):* Permite abrir el archivo seleccionado del cuadro de diálogo.

Atributos:

Nombre atributo	Tipo	Descripción
ag	ExpertAgentMonitor	Se referencia el objeto para utilizar sus métodos y atributos
err	PrintWriter	Maneja los mensajes de error en la salida
fileDir	String	Cadena que contiene el directorio del archivo a abrir
fileName	String	Cadena que contiene el nombre del archivo a abrir
jtaOut	JTextAreaWriter	Se crea objeto para utilizarlo como área de salida
out	PrintWriter	Establece que la salida de impresión será jtaOut
outputArea	JTextArea	Área de texto del panel principal

Clase GetJessParams

Esta clase es parte de la clase ExpertAgentMonitorPanel, y además de tener la función de llamar al cuadro de diálogo de Cargar base para experto..., contiene la llamada a un método definido en la clase ExpertAgentMonitor para que JessAgent lance cada uno de los agentes expertos que conformarán el equipo.

Clase ExpertAgentMonitor

Esta clase extiende la clase Scheduler y permite visualizar la interfaz gráfica generada por la clase ExpertAgentMonitorPanel dentro de la plataforma MadKit y, además, tiene la función de activar a los agentes expertos.

Operaciones:

- *activate():* Método propio de MadKit que es inicialmente llamado cuando el micro-kernel de MadKit registra a un agente. Además de especificar el grupo y el rol del agente registrado, para la aplicación el grupo es *scheduler* y el rol es *expertos*.

- *initGUI():* Método propio de MadKit que se llama para especificar un sistema gráfico externo como el G-Box de MadKit. De esta forma se despliega la interfaz gráfica del panel principal generada por la clase **ExpertAgentMonitorPanel.**

- *lanzar(String Archivo, String fileName):* Se define para lanzar a los agentes expertos representados por una interfaz gráfica que muestran las preguntas contenidas en sus bases de conocimiento. Esta acción se hace por medio del método *launchAgent()* que es propio de MadKit y sus parámetros son: el nuevo agente, el nombre del agente y un valor booleano para establecer al agente.

- *live():* Método propio de MadKit que define el comportamiento principal de los agentes lanzados.

Atributos:

Nombre atributo	Tipo	Descripción
Expertos	Hashtable	Almacena a los agentes expertos activados
display	ExpertAgentMonitorPanel	Establece la salida en la interfaz gráfica

Clase JessAgentMonitorPanel

Esta clase genera la interfaz gráfica de cada agente experto que es lanzado por la clase ExpertAgenMonitor.

Constructor:

JessAgentMonitorPanel(JessAgent _ag, ExpertAgentMonitorPanel_pp), aquí se crea la interfaz para cada uno de los agentes expertos. Esta interfaz contiene un panel de etiquetas de texto, el área de texto para desplegar las preguntas del agente experto y un textfield para que el usuario introduzca las respuestas (sí/no).

Operaciones:

- *actionPerformed(ActionEvent e):* Método propio de MadKit que permite realizar una acción dada mediante la interface con la clase **ActionListener.** En este caso la acción es almacenar la respuesta dada al agente experto (si/no) para que se pueda procesar por RETE.

■ *setFocus():* Método que permite accionar el cursor en el textfield para comenzar a escribir.

Atributos:

Nombre atributo	Tipo	Descripción
Ppal	ExpertAgentMonitorPanel	Permite imprimir la salida en el área definida en ExpertAgentMonitorPanel
Ag	JesAgent	Se referencia el objeto para utilizar sus métodos y atributos
jtaOut	JTextAreaWriter	Se crea objeto para utilizarlo como área de salida
out	PrintWriter	Establece que la salida de impresión será el objeto jtaOut
outputArea	JTextArea	Área de texto para las preguntas
textfield	JTextField	Campo de texto para introducir las respuestas
leer	TextReader	Permite leer las respuestas introducidas por el usuario para ser procesadas

Clase JessAgent

Clase que permite visualizar la base de conocimientos correspondiente a cada agente experto lanzado.

Constructores:

JessAgent(String Archivo, ExpertAgentMonitorPanel _pp), recibe como argumentos el Archivo (que es una cadena con el nombre del archivo a ejecutar).

JessAgent(String Archivo, ExpertAgentMonitorPanel_pp, String f)

Operaciones:

■ *activate():* Método que tiene la función de activar a cada uno de los agentes expertos. Al activar a los agentes, se les debe dar la capacidad de "razonamiento", es decir, se llama a un método que conecta la máquina de inferencia a cada agente. Posteriormente se ejecuta el comando batch de Jess por medio de RETE para cargar el archivo .clp que corresponde a la base de conocimientos, utilizando el método getLoadFile().

- *getLoadFile():* Permite regresar el valor que contiene el atributo loadFile, que es el archivo *.clp que será ejecutado por medio del comando batch de Jess dentro del método activate().

- *initGUI:* Método propio de MadKit que se llama para especificar un sistema gráfico externo como el G-Box de MadKit, y así preparar la interfaz gráfica del agente. De esta forma se visualiza la interfaz gráfica para cada agente experto, misma que se creó en la clase JessAgentMonitorPanel.

- *introRespuesta(TextReader leer, JTextAreaWriter jtaOut):* Método que es llamado en la clase JessAgentMonitorPanel para accionar el textfield en el que serán introducidas las respuestas a cada pregunta que muestren los agentes expertos.

- *isReady():* Método que verifica que cada agente experto esté listo para recibir y enviar mensajes. El mensaje es enviado por cualquier agente experto y el propósito es dar a conocer al resto del equipo a qué nodo de su árbol de decisión se dirige y este mensaje se muestra en la pantalla.

- *know(Value v):* Método que tiene como parámetro el valor de la variable current de la clase Rete. El contenido de esta variable es lo que en cada momento se esté procesando, es decir, el nodo del árbol que corresponde a una pregunta o la respuesta introducida por el usuario que puede ser sí o no.

- *know(String t):* Método que tiene como parámetro el nodo que recibe como mensaje el resto de los agentes. El parámetro es enviado por el Scheduler.

- *println(String s):* Acciona la impresión de datos en la salida tipo PrintWriter.

Atributos:

Nombre atributo	Tipo	Descripción
Ppal	ExpertAgentMonitorPanel	Permite imprimir la salida en el área definida en ExpertAgentMonitorpanel
Display	JessAgentMonitorPanel	Establece la salida en la interfaz gráfica
loadFile	String	Contiene el archivo a cargar por default

Nombre atributo	Tipo	Descripción
Out	PrintWriter	Definición de la salida de impresión
Pw	PrintWriter	Interfaz para introducir la respuesta
Ready	bolean	Define si el agente está listo para recibir un mensaje
Viene	bolean	Recibe el valor de Rete

4.4.2. Diagrama de estados

Este diagrama muestra una secuencia ordenada de eventos (cambios de estados) que ocurren en cada uno de los agentes (clases) que participan en el sistema.

Se presentan 4 diagramas de estados, correspondientes a cada una de las clases implementadas para el framework:

Diagrama de estados de la clase ExpertAgentMonitorPanel (Fig.4.5).

Mediante este diagrama se representa el comportamiento de la clase ExpertAgentMonitorPanel, su función es crear el Panel Principal del SiCAE. Este panel contiene varios elementos como son la barra de menú, la barra de herramientas y el área de texto. El estado inicial es el constructor de la clase, donde se comienzan a crear cada uno de los elementos.

Los elementos de la barra de menú son los siguientes: **Inicializar...,** creado por el método menulnicia_actionPerformed(); el menú **Abrir...,** creado por el método menuConsulta_actionPerformed(); el menú **Limpiar área de texto,** creado por el método menuLimpia-_actionPerformed(); el menú **Ayuda,** creado por el método menuHelp_actionPerformed() y el menú **Salir,** creado por el método menuSalir_actionPerformed(). Una vez creado la barra de menús, se crea la barra de herramientas que contiene las mismas opciones.

Los elementos de la barra de herramientas son: botón **Inicializar experto,** creado por el método buttonAbrir_actionPerformed(); botón **Abrir archivo,** creado por el método buttonConsulta_actionPerformed(); botón **Limpiar** área de texto, creado por el método buttonLimpiar_actionPerformed(); botón **Ayuda,** creado por el métodobuttonAyuda_actionPerformed() y el botón **Salir,** creado por el método buttonSal¡r_actionPerformed().

Diagrama de estados de la clase ExpertAgentMonitor (Fig.4.6).

Este diagrama representa la funcionalidad de la clase ExpertAgentMonitor, que es la de activar a cada uno de los agentes expertos con su respectiva interfaz gráfica, así como controlar su ciclo de vida y activar el grupo al que pertenecen.

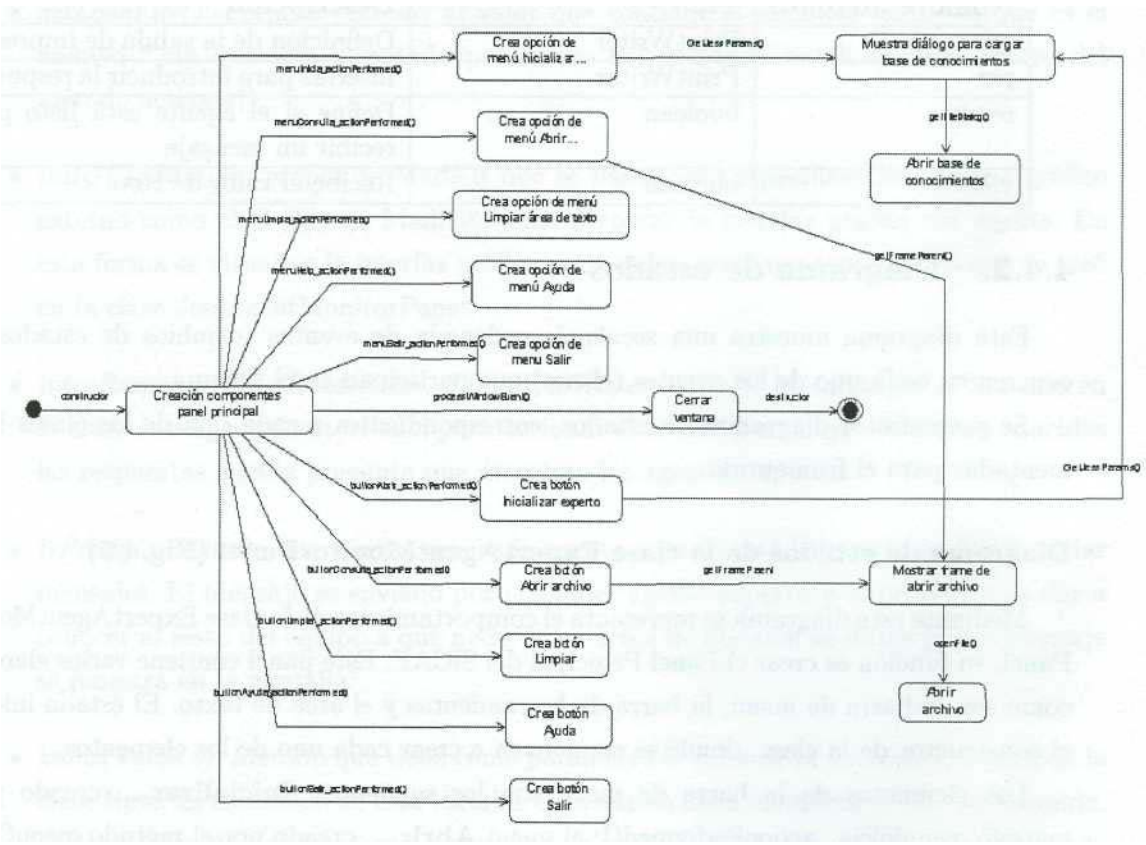

Figura 4.5: *Diagrama de estados correspondiente a la clase* ***ExpertAgentMonitorPanel***

La secuencia de los estados es la siguiente: Se activa la interfaz gráfica para cada agente experto mediante el método initGUI(), después se determina el grupo y roles de los agentes por medio del método activate(), posteriormente se debe tener un control del ciclo de vida para cada agente que se activa, esto lo hace el método live(). Una vez determinado lo anterior, se puede lanzar cada agente experto a través del método lanzar().

Diagrama de estados de la clase JessAgentMonitorPanel (Fig.4.7).

El comportamiento de la clase JessAgentMonitorPanel, es el de crear la interfaz gráfica para cada agente experto, que es mostrada por le clase ExpertAgentMonitor.

La secuencia de estados es la siguiente: a través del constructor se crean los elementos de

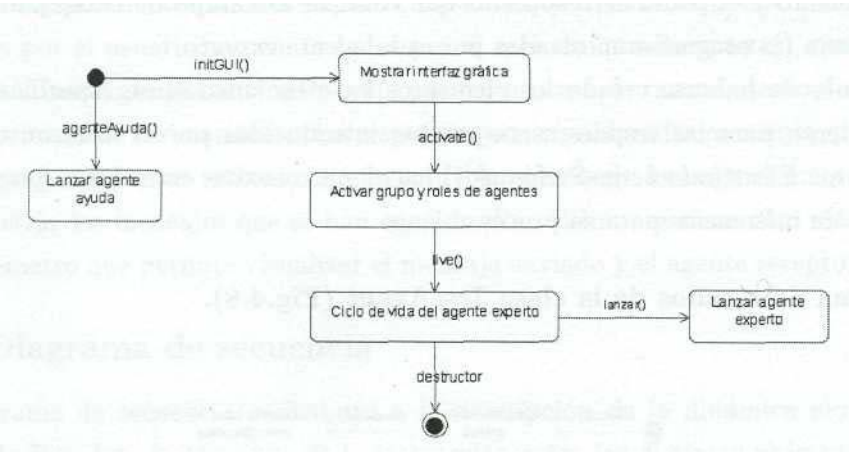

Figura 4.6: *Diagrama de estados correspondiente a la clase* ***ExpertAgentMonitor***

Figura 4.7: *Diagrama de estados correspondiente a la clase* ***JessAgentMonitorPanel***

la interfaz gráfica como son el área de texto para las preguntas, el panel de etiquetas para cada elemento y el panel de respuestas que contiene el campo de texto para introducir las respuestas a las preguntas mostradas por cada agente experto.

Después de haberse creado los elementos ya mencionados, se especifica el proceso de razonamiento para las respuestas respuestas introducidas por el usuario, que pueden ser dos: sí o no. El método actionPerformed() hace una conexión entre las respuestas dadas y la máquina de inferencias para su procesamiento.

Diagrama de estados de la clase JessAgent (Fig.4.8).

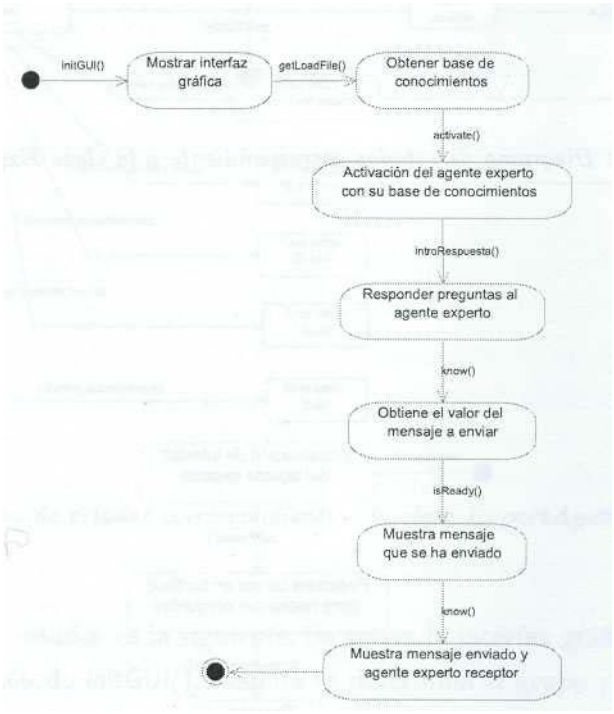

Figura 4.8: *Diagrama de estados correspondiente a la clase* ***JessAgent***

El comportamiento de esta clase está relacionada con la clase JessAgentMonitorPanel, su funcionamiento es cargar la base de conocimiento para cada agente experto, así como procesarla. El método initGUI(), permite mostrar la interfaz gráfica creada por la clase **Jess-AgentMonitorPanel,** para cargar la base de conocimiento correspondiente se usa el método getLoadFile(), cuando se ha seleccionado la base de conocimiento se activa el agente experto

con el método activate(), en este momento el usuario puede visualizar cada interfaz con las preguntas correspondientes a cada experto, mismas que deberá responder. Las respuestas introducidas por el usuario son controladas por el método introRespuesta(), que es llamado desde la clase **JessAgentMonitorPanel()** para el proceso de razonamiento de las mismas.

Durante el proceso de razonamiento, las respuestas son consideradas mensajes que son enviados a los agentes expertos activos, esto lo controla el método know(); el método isReady() permite mostrar los mensajes que se han enviado. Por último, se tiene otro método know() con un parámetro que permite visualizar el mensaje enviado y el agente receptor del mismo.

4.4.3. Diagrama de secuencia

El Diagrama de secuencia contribuye a la descripción de la dinámica mostrada en el Diagrama de Estados, en términos de la interacción entre los distintos objetos del sistema. Tal interacción se lleva a cabo a través de mensajes [Rendón, 2000].

Este diagrama (Fig. 4.9) se obtiene a partir del Diagrama de Estados de cada agente.

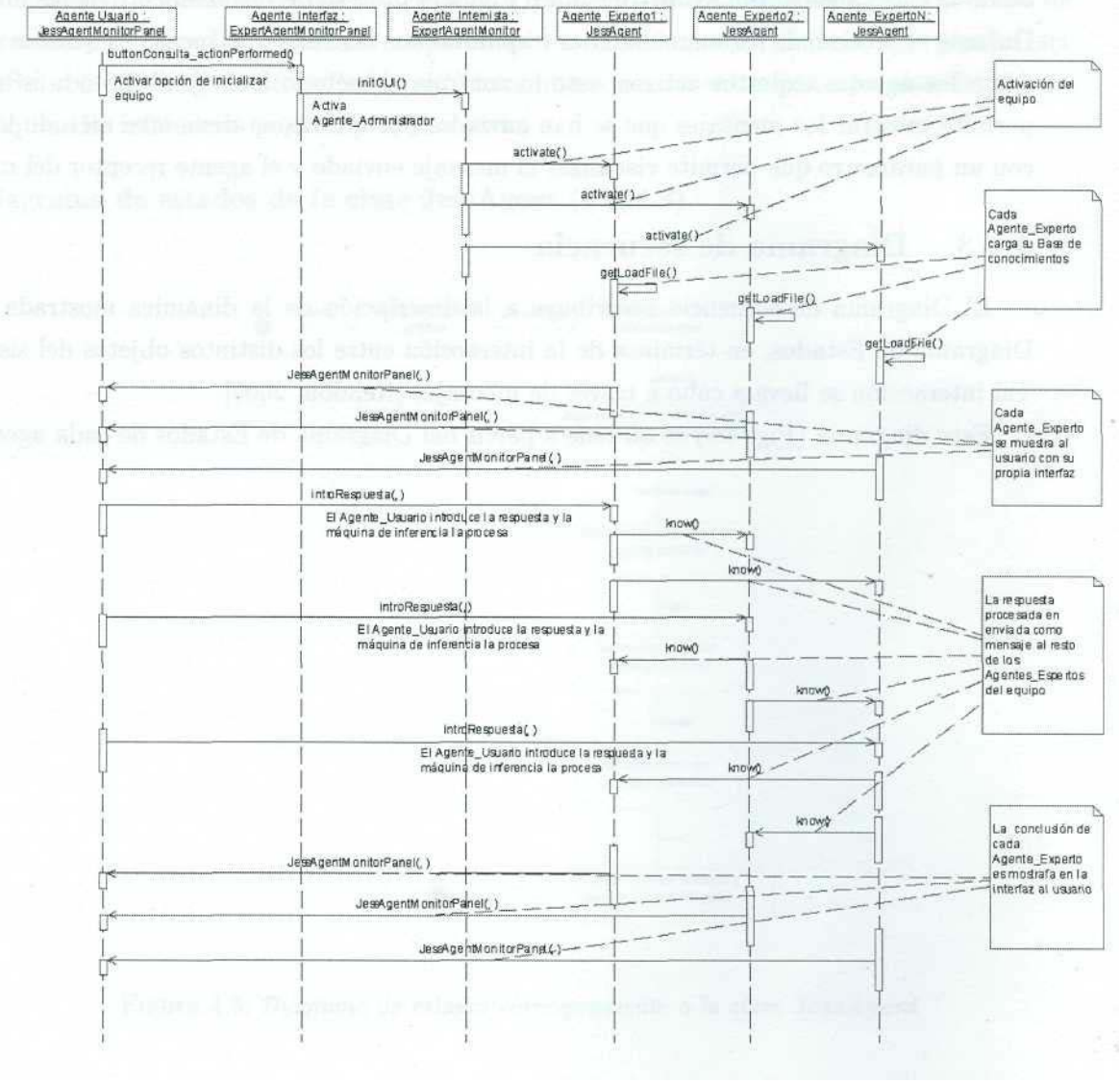

Figura 4.9: *Diagrama de secuencia del framework desarrollado.*

Descripción del Diagrama de secuencia:

1. El usuario inicializa la consulta. El método que recibe la acción que el usuario ha iniciado es actionPerformed().

2. Se inicializan los agentes expertos que formarán el equipo. El método initGUl() permite mostrar el cuadro de diálogo correspondiente a cada agente experto. El número de agentes expertos que estarán activos en el equipo, dependerá del usuario y el problema.

   Cabe mencionar que en el diagrama se representaron "N" número de agentes expertos, con la finalidad de establecer que el número de éstos no está definido.

3. El agente experto inicializado cargará su base de conocimientos correspondiente mediante el método getLoadFile().

4. Los agentes expertos activos en el equipo formado, comenzarán a consensar con la información proporcionada por el usuario para poder llegar a un acuerdo y una conclusión aceptable.

   En el diagrama se representa la colaboración arbitraria entre agentes expertos, es decir, la colaboración no está restringida ni por el usuario, ni por el número de agentes expertos existentes.

5. El usuario podrá entonces responder las preguntas desplegadas por cada agente experto.

   Dependiendo del problema, el usuario podrá intervenir con el equipo de agentes expertos para proporcionar su propia opinión y llegar a una solución conjunta, por ello se ha definido al usuario como un *agente usuario.*

4.4.4. Diagrama de colaboración

El diagrama de colaboración no solamente muestra los mensajes a través de los cuales se produce la interacción entre los objetos, como en el diagrama de secuencia, sino también los enlaces entre los objetos. El énfasis de un diagrama de colaboración está en la estructura formada por los objetos y sus enlaces [Rendón, 2000].

En la Fig.4.10, se muestra el Diagrama de Colaboración de la aplicación.

Descripción del Diagrama de colaboración:

Los mensajes entre los objetos están numerados de forma secuencial, lo que permite visualizar la interacción general de todos los objetos existentes. A continuación se describe la secuencia de mensajes:

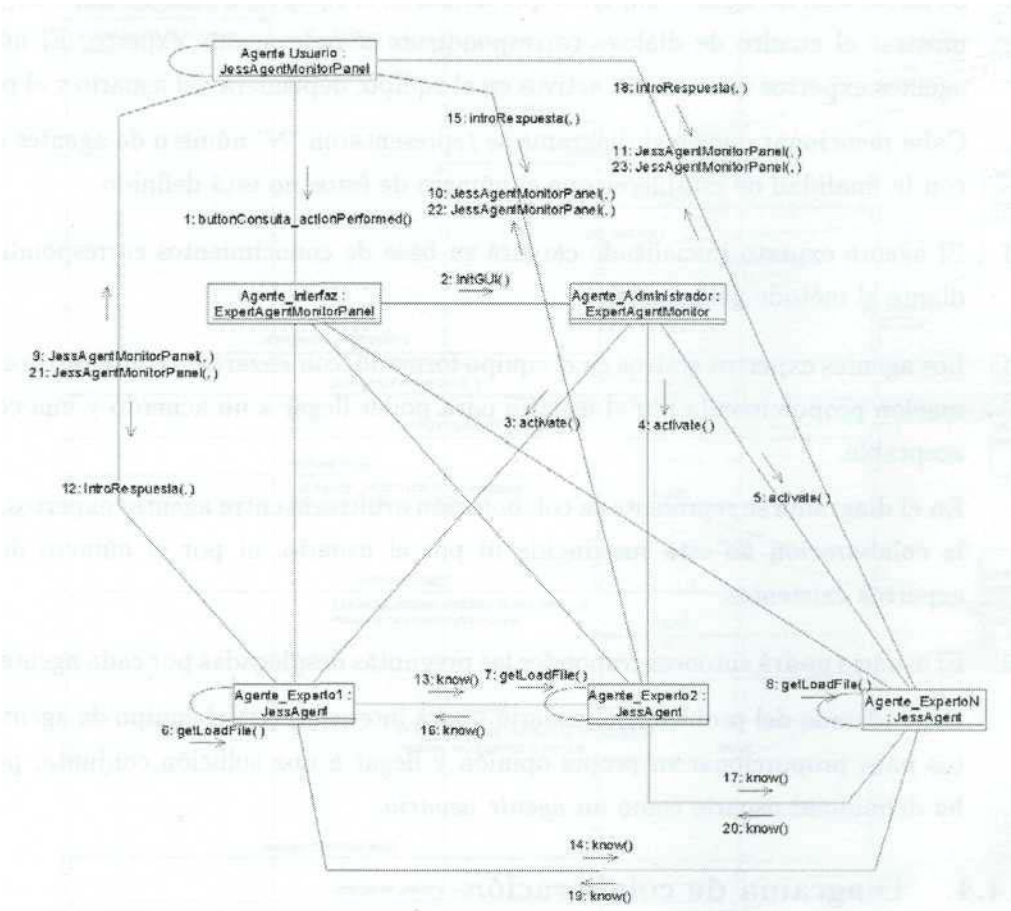

Figura 4.10: *Diagrama de Colaboración de la aplicación.*

- El Agente_Usuario a través del método l:buttonConsulta_actionPerformed(), se activa el Agente _Interfaz.
- El Agente__Interfaz mediante el método 2:initGUI() activa al Agente_Administrador, que es el encargado de activar al equipo de Agentes_Expertos.

- El Agente_Administrador activa a los integrantes del equipo de expertos: 3:activate(), 4:activate(), 5:activate().

- Cuando el equipo de expertos ha sido activado, cada uno de ellos cargará su propia base de conocimiento: 6:getLoadFile(), 7:getLoadFile(), 8:getLoadFile().

- Cada Agente_Experto, tiene un enlace con el Agente_Usuario: 9:JessAgentMonitor-Panel(), 10:JessAgentMonitorPanel(), H:JessAgentMonitorPanel(), a través de este en lace, el usuario podrá contestar las preguntas: 12:introRespuesta(), 15:introRespuesta(), 18:introRespuesta().

- De acuerdo a la secuencia de mensajes, puede observarse que en el momento que el Agente_Usuario introduce la respuesta para el Agente_Experto 1, éste envía mensaje a los agentes Agente_Experto2 y Agente_ExpertoN mediante 13:know() y 14:know() respectivamente. Así también cuando el Agente_Usuario responde a la pregunta hecha por el Agente_Experto2, éste envía mensaje al Agente_Expertol y Agente__ExpertoN con 16:know() y 17:know() respectivamente. El mismo procedimiento se hace cuando el Agente_Usuario responde al Agente_ExpertoN, ya que éste envía mensajes al Agente_Expertol y Agente_Experto2 mediante: 19:know() y 20:know().

- Se tiene un enlace de regreso entre el Agente_Usuario y cada Agente_Experto, ya que las respuestas y mensajes enviados son mostrados en la interfaz, los enlaces son: 21:JessAgentMonitorPanel(), 22:JessAgentMonitorPanel(),23:JessAgentMonitorPanel().

4.5. Implementación del SiCAE

Al concluir las etapas de Análisis y Diseño, se procede a la implementación del framework. La Fig. 4.11, muestra el panel principal del prototipo de framework desarrollado, con las siguientes opciones:

- ***Inicializar experto:*** Permite al usuario lanzar a todos los agentes expertos que formarán el equipo. Opcicm que también puede ser ejecutada mediante el menú *Agentes*. El conocimiento que tengan los agente activados, dependerá del dominio de aplicación del caso de estudio particular.

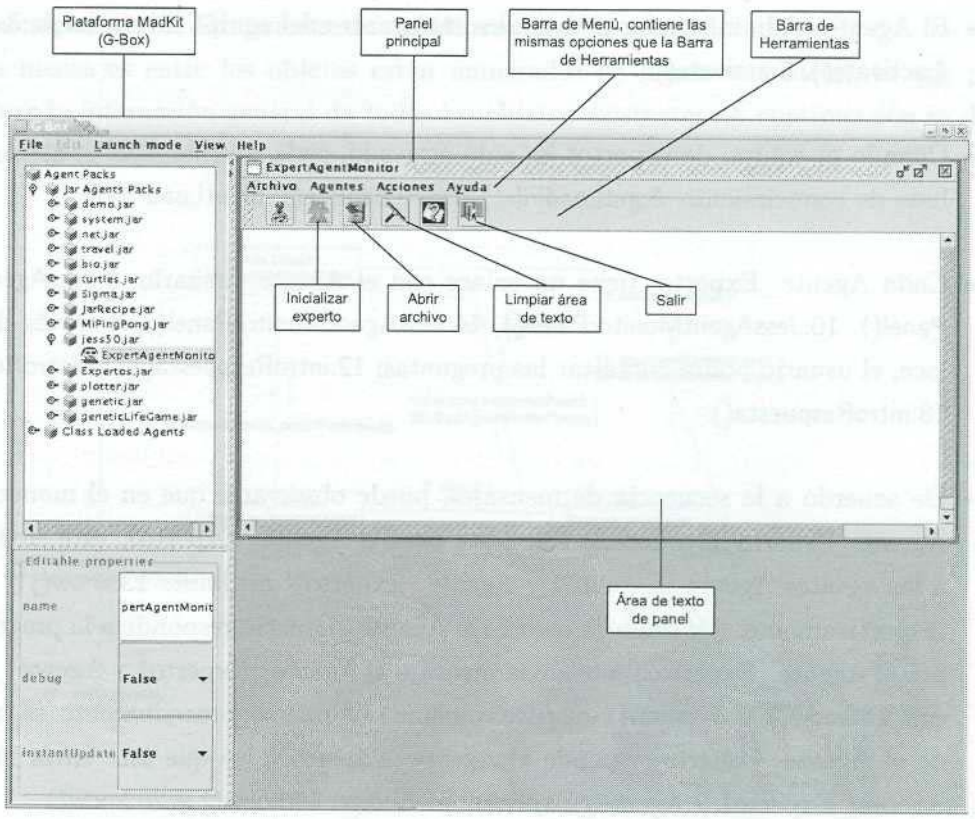

Figura 4.11: *Panel principal del SiCAE.*

- ***Abrir archivo:*** Da al usuario la posibilidad de abrir algún archivo de tipo texto o .clp[1] para su visualización. Opción disponible en el menú *Archivo.*

- ***Limpiar área de texto****:* Con este botón se limpia el área de texto del panel principal, en caso de haber activado la opción de Abrir archivo. Opción disponible en el menú *Acciones.*

- ***Salir:*** Esta opción termina completamente la aplicación y el ambiente MadKit. Opción disponible en el menú *Archivo.*

En la Fig. 4.12, se observa un equipo de dos agentes expertos que despliegan las preguntas de sus bases de conocimientos.

[1] Extensión de archivo utilizada por las bases de conocimiento de Jess.

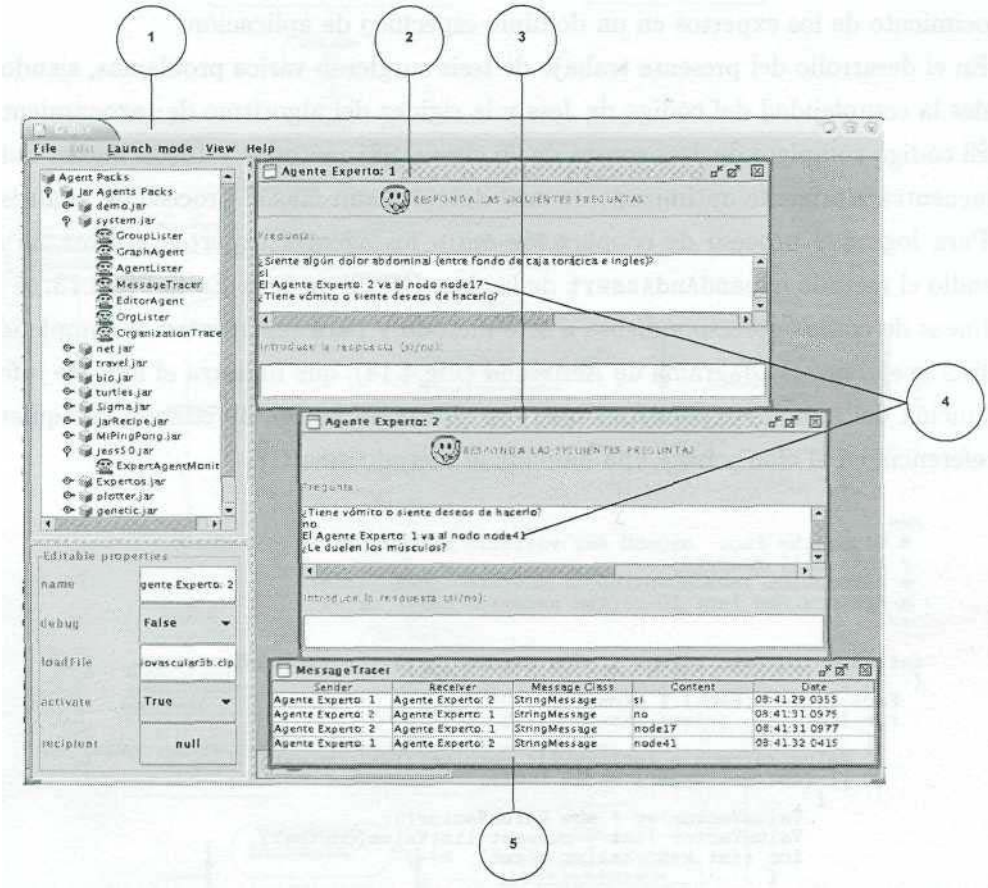

Figura 4.12: *Ejecución de la aplicación con un equipo de dos agentes expertos.*

En esta figura se observan los siguientes puntos: el número 1 indica el ambiente Gbox donde se ejecuta la aplicación. Los puntos 2 y 3, representan al equipo de Agentes Expertos activado, el número de éstos y su conocimiento, dependerán del caso de estudio específico. El número 4 apunta a la interfaz de cada Agente Experto, donde el usuario podrá visualizar las preguntas hechas por los agentes expertos, introducir sus respuestas y visualizar algún tipo de mensaje que el experto envíe. El punto número 5, es un visor de mensajes propio de la plataforma MadKit, donde se visualiza el agente que envía el mensaje, el agente que lo recibe, tipo de mensaje, el contenido del mensaje y la hora en la que se envió dicho mensaje.

El framework desarrollado, se considera de tipo general, ya que dependiendo del caso de estudio, se desarrollarán las bases de conocimientos correspondientes, que representarán el

conocimiento de los expertos en un dominio específico de aplicación.

En el desarrollo del presente trabajo de tesis surgieron varios problemas, siendo los principales la complejidad del código de Jess y la rigidez del algoritmo de razonamiento RETE.

El código completo de Jess consta de 76 clases, 981 métodos y 19935 líneas. Este código, se encuentra altamente optimizado, lo cual dificulta aún mas el proceso de análisis.

Para lograr el proceso de cooperación entre los agentes expertos activos, se analizó y extendió el método expandAndAssert de la clase RETE de Jess. En la Fig.4.13, se muestran las líneas de código correspondiente a este método y para representar la complejidad de su análisis se elaboró el Diagrama de Actividad (Fig.4.14), que muestra el flujo de información. La función del método expandAndAssert es clonar un hecho, heredando cualquier variable de referencia en el clon y hace una llamada al método assert().

```
/**
 * Clone ths fact, expand any variable refersnees in the clone,
 * then cali assert().
 * @exception JessException If anything goes wrong.
 * @return The fact ID of the asserted fact, or —1.

*/

int expandAndAssert(Fact f, Context context) throws JessException
{
  Fact fact = (Fact) f.clone();
  for (int j = 0; j < fact . size(); j++)
   {
       Value current = fact.get(j).resolveValue(context);
       if (current.type() == RU.LIST)
       {
           ValueVector vv = new ValueVector();
           ValueVector list = current.listValue(context);
           for (int k=0; k<list.size(); k++)
           {
               Value listItem - list.get(k).resolveValue(context);
               if (listItem.type() == RU.LIST)
               {
                   ValueVector sublist = listItem.listValue(context);
                   for (int m=0; m<sublist.size(); m++)
                     vv.add(sublist.get(m).resolveValue(context));
               }
               else
                 vv.add(listItem);
            }
        current = new Value(vv. RU.LIST);
       }
       fact.set(current, j) ;
     }
  return assert(fact);
```

Figura 4.13: *Líneas de código del método expandAndAssert de la clase Rete.*

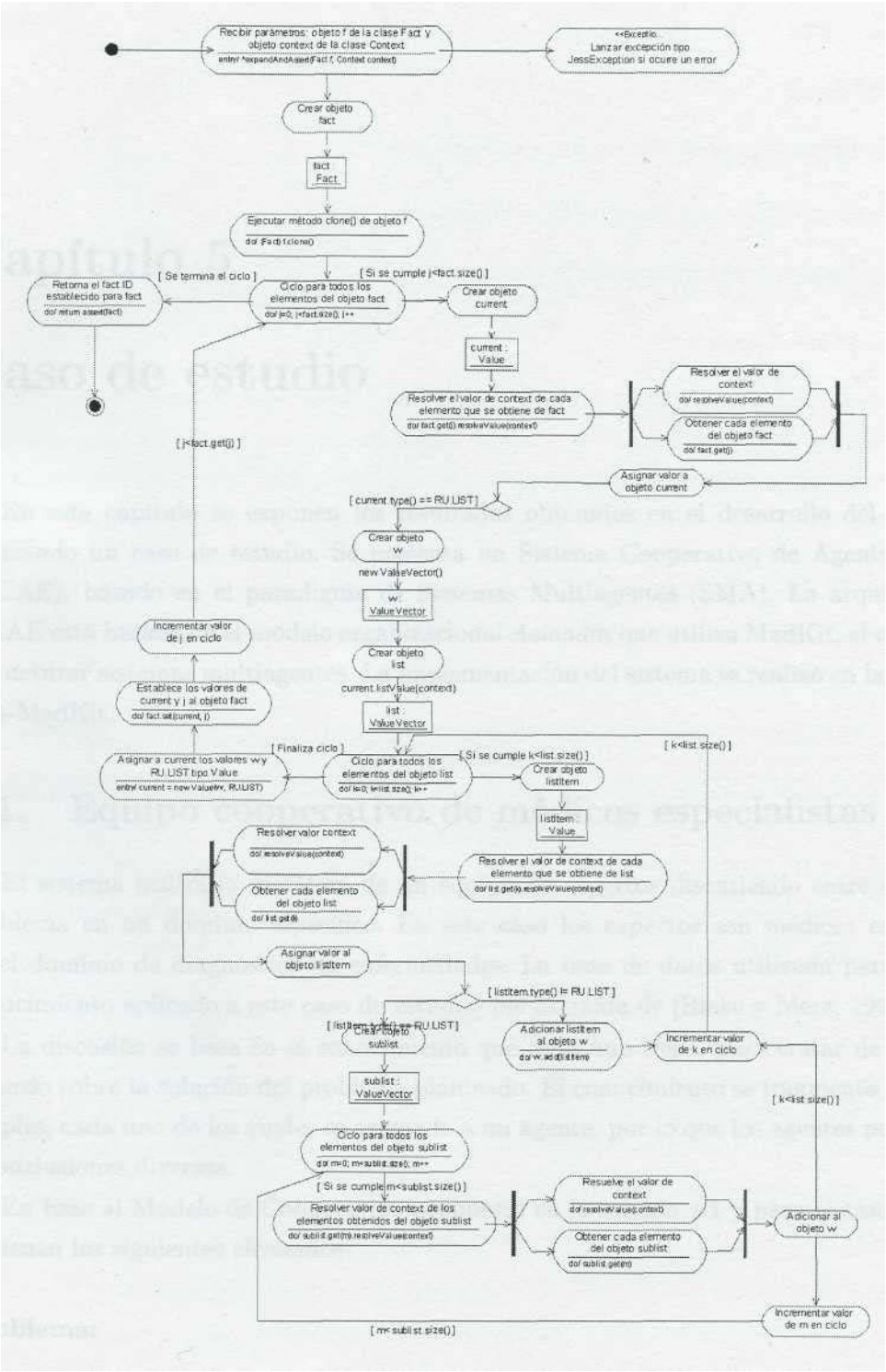

Figura 4.14: *Diagrama de actividad del método expandAndAssert de la clase Rete (Jess)*

Capítulo 5

Caso de estudio

En este capítulo se exponen los resultados obtenidos en el desarrollo del framework, utilizando un caso de estudio. Se presenta un Sistema Cooperativo de Agentes Expertos (SICAE), basado en el paradigma de Sistemas Multiagentes (SMA). La arquitectura del SICAE está basada en el modelo organizacional *Aalaadin* que utiliza MadKit, el cual permite estructurar sistemas multiagentes. La implementación del sistema se realizó en la plataforma Jess-MadKit.

5.1. Equipo cooperativo de médicos especialistas

El sistema utiliza la metáfora de un equipo de expertos discutiendo entre sí, sobre un problema en un dominio específico. En este caso los expertos son médicos especialistas, en el dominio de diagnóstico de enfermedades. La base de datos utilizada para extraer el conocimiento aplicado a este caso de estudio, fue extraída de [Blake y Merz, 1998].

La discusión se basa en el conocimiento que cada uno tiene, para tratar de llegar a un acuerdo sobre la solución del problema planteado. El conocimiento se fragmenta en módulos simples, cada uno de los cuales es asignado a un agente, por lo que los agentes pueden llegar a conclusiones diversas.

En base al Modelo de Cooperación propuesto en la sección 4.1 y para el caso de estudio se tienen los siguientes elementos:

Problema:

Se diagnosticará una enfermedad de acuerdo a las respuestas dadas por los agentes expertos que forman un equipo.

Subproblemas:

- **AE_1** diagnosticará enfermedades gastrointestinales.
- **AE_2** diagnosticará enfermedades neurológicas.
- **AE_3** diagnosticará enfermedades cardiovasculares.

Asignación de Tareas:

1. A cada AE_i se le asignará el conocimiento, que representa el grado de experiencia de cada uno de ellos. Cada agente experto activo en la aplicación tienen asignada una BCj, que serán procesadas internamente por el Algoritmo RETE y están representadas por medio de árboles de nodos de decisión.

- Al AE_1 se le asigna BC_1, que contiene el conocimiento codificado de las enfermedades gastrointestinales, en la Fig.5.1 se presenta una parte de la base de conocimiento de este agente.

```
(node (nameroot) (type decision) (question "¿Siente algún dolor abdominal (entre fondo de caja torácica e ingles)?") (yes-node node41) (no-node node93) (answer nil) (certainty nil))
(node (name node41) (type decision) (question "¿Tiene vómito o siente deseos de hacerlo?") (yes-node node37) (no-node node43) (answernil) (certainty nil))
(node (name node37) (type decision) (question "¿Furna?") (yes-node node90) (no-node node90) (answer nil) (certainty nil))
(node (name node90) (type decision) (question "¿Ha observado sangre en su vómito o deposiciones?") (yes-node node91) (no-node node92) (answer nil) (certainty nil))
(node (namenode91) (type decision) (question "¿Tiene falta de apetito con repugnancia a los alimentos?") (yes-node node43) (no-node node87) (answer nil) (certainty nil))
(node (name node92) (type decision) (question "¿Tiene mal aliento o la lengua sucia?") (yes-node node91) (no-node node43) (answer nil) (certainty nil))
(node (name node43) (type decision) (question "¿Sufre de diarreas?") (yes-node node8) (no-node node97) (answer nil) (certainty nil))
(node (name node8) (type decision) (question "¿Tiene fiebre (por encima de 38 grados)?") (yes-node node86) (no-node node 116) (answer nil) (certainty nil))
(node (name node86) (type decision) (question "¿Presenta dolor abdominal (parte inferior izquierda)?") (yes-node node95) (no-node node 87) (answer nil) (certainty nil))
(node (name node87) (type decision) (question "¿Presenta dolor abdominal (ombligo y lado inferior derecho)?") (yes-node node62) (no-nodenode 16) (answer nil) (certainty nil))
(node (name node95) (type decision) (question "¿Tiene inflamación de abdomen con gases?") (yes-node node96) (no-node gen 14) (answer nil) (certainty nil))
(node (name node96) (type decision) (quesnon "¿Tiene algún bulto anormal en su abdomen?") (yes-node node49) (no-node gen 14) (answer nil) (certainty mi))
(node (name node49) (type decision) (question "¿Sufre de estreñimiento?") (yes-node node 17) (no-node gen 14) (answer nil) (certainty nil))
(node (name node 17) (type decision) (question "¿Le duelen los músculos?") (yes-node nodeí 16) (no-node gen2) (answer Enfermedad_de_crohn) (certainty 0.57))
(node (name gen2) (type answer) (question mi) (yes-node nil) (no-node nil) (answer Enfermedad_de_crohn) (certainty 0.57))
```

Figura 5.1: *Una parte de la base de conocimiento de enfermedades gastrointestinales.*

Para que el algoritmo Rete procese la información almacenada en la base de conocimiento, utiliza un árbol de decisiones, el correspondiente a las enfermedades gastrointestinales se muestra en la Fig.5.2.

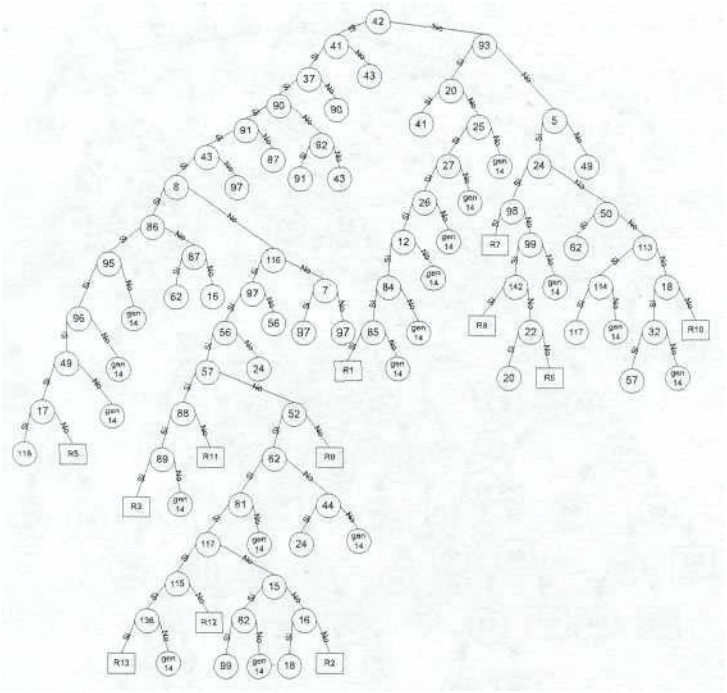

Figura 5.2: *Árbol de nodos para el agente experto de enfermedades gastrointestinales.*

- Al AE2 se le asigna BC2, correspondiente al conocimiento codificado de las enfermedades neurológicas, una parte de la base de conocimiento de este agente, se ilustra en la Fig.5.3.

```
(node (name root) (type decision) (question "¿Sufre sensación de vértigo?") (yes-node node109) (no-node node2) (answer nil) (certainty nil))
(node (name node 109) (type decision) (question "¿Tiene el pulso más lento de lo normal?") (yes-node node41) (no-node node51) (answer nil) (certainty nil))
(node (name nodc41) (type decision) (question "¿Vomita o siente deseos de hacerlo?") (yes-node node34) (no-node gen 14) (answer nil) (certainty nil))
(node (name node34) (type decision) (question" ¿Tiene algún síntoma que ha estado presente por 6 semanas o más"") (yes-node node20) (no-node node28) (answer nil) (certainty nil))
(node (name node20) (type decision) (question"¿Tiene síntomas que se produzcan en ataques continuamente?") (yes-node nodel20) (no-node node7) (answer nil) (certainty nil))
(node (name node 120) (type decision) (question "¿Tiene zumbidos de oidos?") (yes-node gen2) (no-node genl4) (answer Espondilosis_cervical) (certainty 0.80))
(node (name gen2) (type answer) (question nil) (yes-node nil) (no-node nil) (answer Espondilosis_cervical) (certainty 0.80))
(node (name node51) (type decision) (question "¿Tiene pérdida de voz en pequeña cuantía o totalmente?") (yes-node node58) (no-node node54) (answer nil) (certainty nil))
(node (name node58) (type decision) (question "¿Tiene sensanción de desmayo repentina, debilidad o pérdida de conciencia?") (yes-node nodc61) (no-node node63) (answer nil) (certainty nil))
(node (name node61) (type decision) (question "¿Tiene alguna parte de su cuerpo entumecida o siente pinchazos?") (yes-node nodc28) (no-node genl4) (answer nil) (certainty nil))
(node (name node28) (type decision) (question "¿Está confuso acerca de lo que le está ocurriendo?") (yes-node node54) (no-node gen3) (answer Hemorragia_subdural) (certainty 0.66))
(node (name gen3) (type answer) (question nil) (yes-node nil) (no-node nil) (answer Hemorragia_subdural) (certainty 0.66))
```

Figura 5.3: *Una parte de ¡a base de conocimiento de enfermedades neurológicas.*

El Algoritmo Rete hace uso del árbol de nodos que se muestra en la Fig.5.4 para llevar a cabo el proceso de razonamiento.

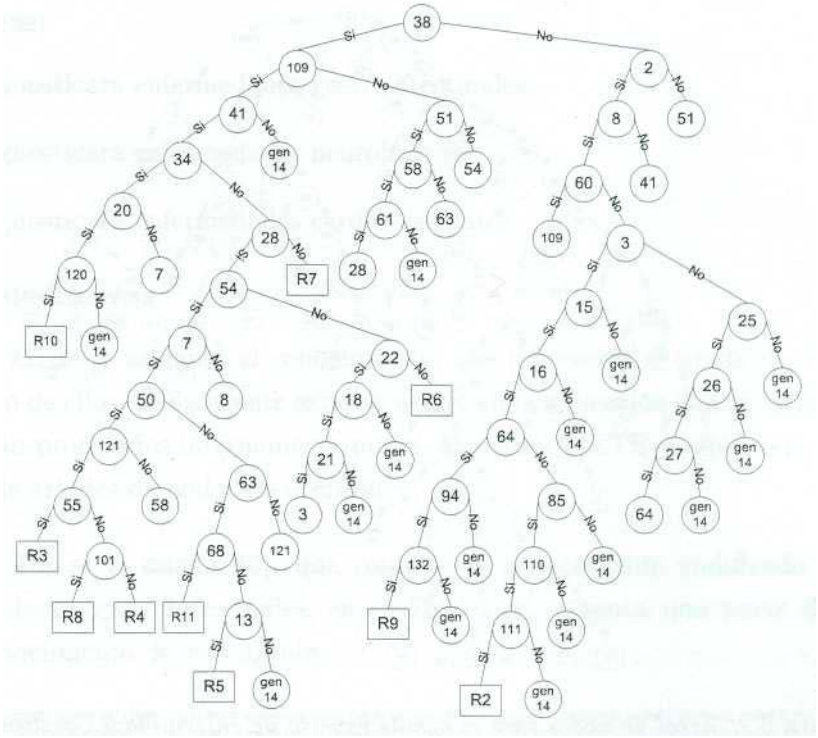

Figura 5.4: *Árbol de nodos para el agente experto de enfermedades neurológicas.*

- Al AE_3 se le asigna BC_3, que contiene el conocimiento codificado de las enfermedades cardiovasculares, en la Fig.5.5 se observa una parte de la base de conocimiento de este agente.

```
(node (name root) (type decision) (question "¿Siente algún dolor abdominal (entre fondo de caja torácica e ingles)?") (yes-node node41) (no-node node8) (answer nil) (certainty nil))
(node (name node41) (type decision) (question "¿Tiene vómito o siente deseos de hacerlo?") (yes-node node 18) (no-node node48) (answer nil) (certainty nil))
(node (name nodel8) (type decision) (question "¿Siente algún dolor en el tórax (pecho)?") (yes-node node57) (no-node nodel6) (answer nil) (certainty nil))
(node (name node57) (type decision) (question "¿Efectúa la deposición con gran cantidad de gases?") (yes-node node46) (no-node gen 14) (answer nil) (certainty nil))
(node (name node46) (type decision) (question "¿Está un poco tenso y aprensivo?") (yes-node node20) (no-node gen14) (answer nil) (certainty nil))
(node (name node 16) (type decision) (question "¿Tiene alteraciones en la deglución?") (yes-node node62) (no-node node63) (answer nil) (certainty nil))
(node (name node62) (type decision) (question "¿Tiene peso excesivo?") (yes-node node52) (no-node node 15) (answer nil) (certainty nil))
(node (name node52) (type decision) (question "¿Sangra con las deposiciones?") (yes-node node56) (no-node node 15) (answer nil) (certainty nil))
(node (name node56) (type decision) (question "¿Ha tenido deposiciones normales recientemente?") (yes-node node81) (no-node gen 14) (answer nil) (certainty nil))
(node (name node81) (type decision) (question "¿Tiene laxitud, falta de apetito y ha perdido peso?") (yes-node node90) (no-node node47) (answer nil) (certainty nil))
(node (name node90) (type decision) (question "¿Observa sangre en su vómito o deposiciones (rojao negra)?") (yes-node node91) (no-node node23) (answer mi) (certainty nil))
(node (name node91) (type decision) (question "¿Tiene falta de apetito con repugnancia a los alimentos?") (yes-node gen2) (no-node gen 14) (answer Fibrilación_auricular) (certainty 0.55))
(node (name gen2) (type answer) (question nil) (yes-node nil) (no-node nil) (answer Fibrilación_auricular) (certainty 0.55))
```

Figura 5.5: *Una parte de la base de conocimiento de enfermedades cardiovasculares.*

El Algoritmo Rete hace uso del árbol de nodos que se ilustra en la Fig.5.6, para realizar el proceso de razonamiento.

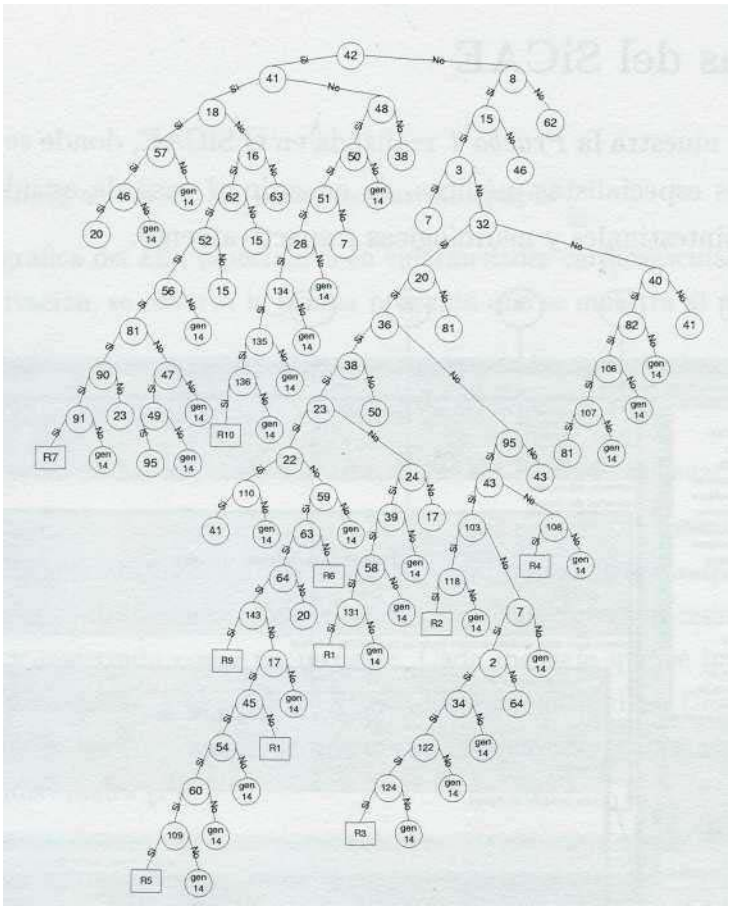

Figura 5.6: *Árbol de nodos para el agente experto de enfermedades cardiovasculares.*

2. Se tienen otros agentes con roles específicos, que ayudarán a alcanzar el objetivo del SICAE.

- Ain: El Agente de interfaz tiene la tarea de mantener la comunicación entre el
  usuario y el resto de los elementos del sistema.

- Aad: El Agente administrador tiene las tareas de: coordinación y control dentro
  del sistema, integración y obtención de resultados globales.

El framework implementado, hace uso de la plataforma MadKit, específicamente a través de una plataforma estándar GUI (Graphic User Interface) llamada G-Box, que permite la ejecución y desarrollo de sistemas de agentes.

5.2. Pruebas del SiCAE

En la Fig. 5.7, se muestra la *Prueba 1* realizada en el SiCAE, donde se activa a un equipo de Agentes Expertos especialistas médicos, de acuerdo al caso de estudio: especialistas en enfermedades gastrointestinales y neurológicas respectivamente.

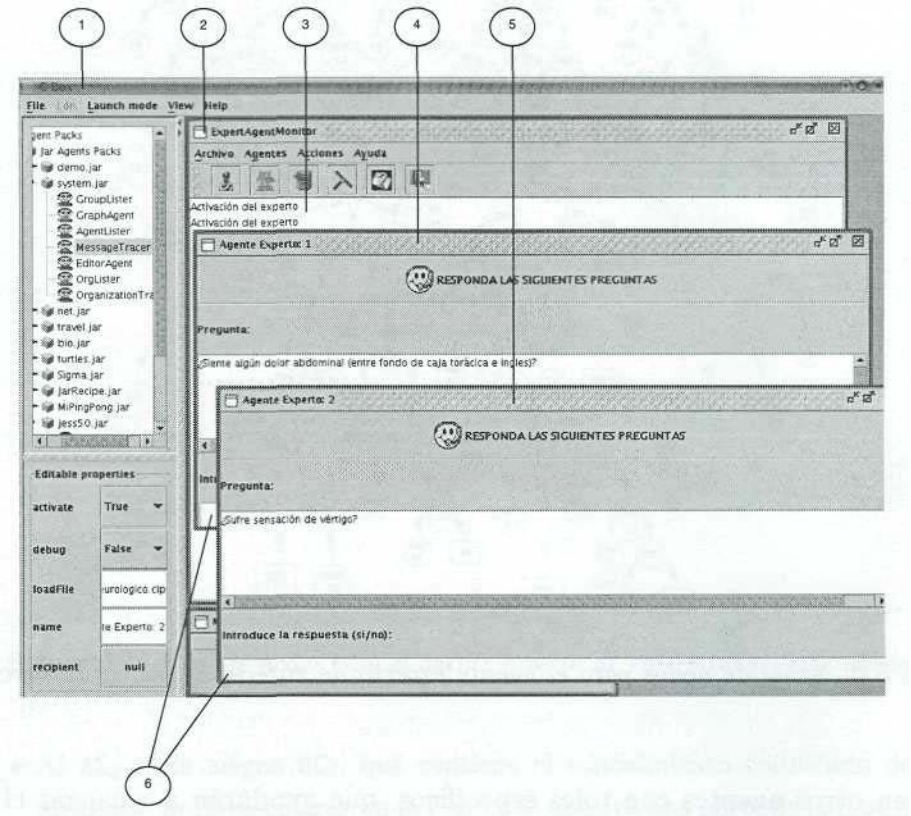

Figura 5.7: *Activación de un equipo de dos agentes expertos, dominio de aplicación de enfermedades.*

A continuación se describen los resultados de la *Prueba 1,* de acuerdo a la numeración de la Fig.5.7:

1. Indica la interfaz gráfica de la plataforma G-box de MadKit, en este caso permite la visualización del framework desarrollado (SiCAE).

2. Interfaz gráfica del Panel Principal del SiCAE.

3. El Panel Principal del SiCAE tiene una área de texto, que permite visualizar algunos mensajes, en este caso muestra el mensaje "Activación de experto", lo hace cada vez

que el usuario activa un AE_i para integrar un equipo.

4. Interfaz gráfica del AE_1; especialista en enfermedades cardiovasculares. En el momento de la activación, se observa la primer pregunta que se muestra al usuario.

5. Interfaz gráfica del AE_2, especialista en enfermedades neurológicas. Se observa la pregunta que se hace al usuario al momento de su activación.

6. Cada interfaz de los AE_i tiene un campo de texto, donde el usuario introduce su respuesta.

Después de la activación del equipo de AE_i, el usuario procederá a responder las preguntas correspondientes. Cada respuesta que el usuario introduce, es procesada por la máquina de inferencias y registrada como un mensaje. Cada mensaje que se intercambia entre los AE_i activos, representa la cooperación que se establece entre ellos. Después de que ambos agentes han concluido con la sesión de preguntas y respuestas, ambos agentes mostrarán sus conslusiones individuales (Fig.5.8).

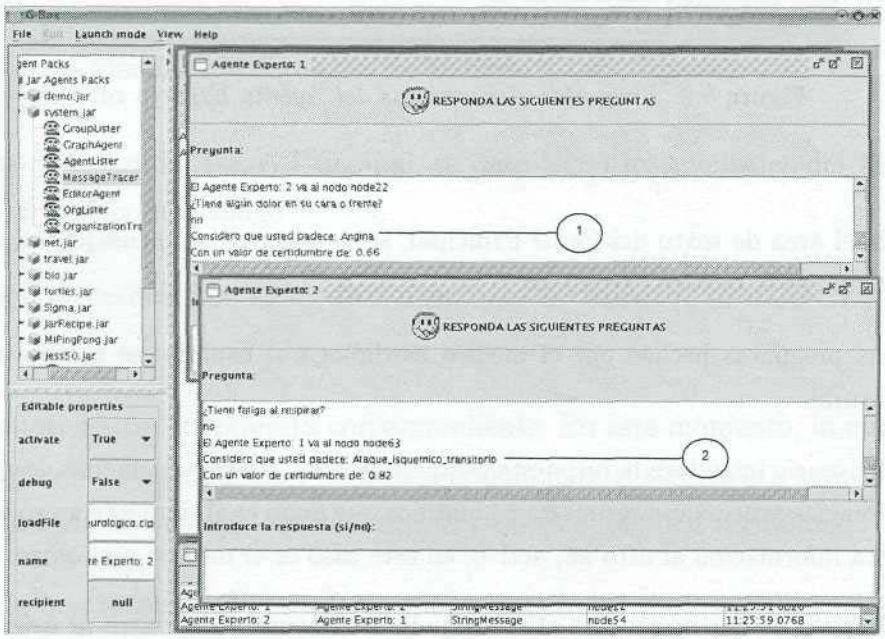

Figura 5.8: *Conclusiones generadas por cada Agente Experto.*

A continuación se muestra el conjunto de preguntas, respuestas y mensajes para cada uno de los AE_i activos.

En la Fig.5.9, se visualiza lo siguiente de acuerdo a las etiquetas:

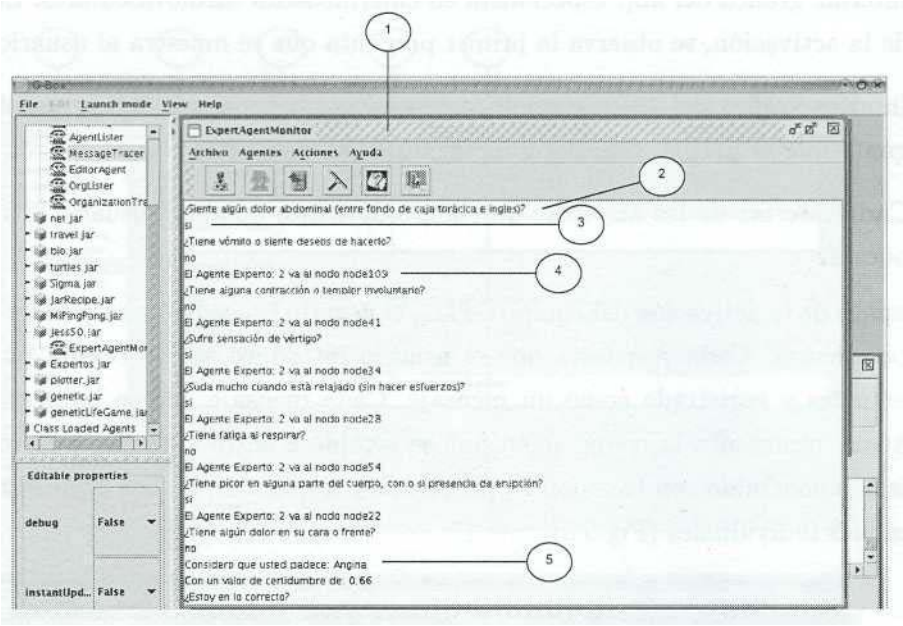

Figura 5.9: *Preguntas y respuestas del Agente Experto cardiólogo.*

1. En el área de texto del Panel Principal, se despliegan los mensajes del $AE_{1;}$ que en el equipo es el médico cardiólogo.

2. Las preguntas hechas por el médico cardiólogo al usuario, se muestran en lenguaje natural.

3. El usuario introduce la respuesta correspondiente. En este momento, la máquina de inferencias procesa esa respuesta e identifica qué nodo es el siguiente a procesar, enviando esta información al otro AE_i activo, en este caso es el médico neurólogo.

4. Después de que se procesa el mensaje internamente, se muestra al usuario, indicando qué nodo será procesado, es decir, qué pregunta será la que despliegue el otro AE_3 activo en su interfaz, para que el usuario la conteste.

5. Concluido el proceso de razonamiento, el AE_i es capaz de dar su conclusión de acuerdo a las respuestas introducidas por el usuario.

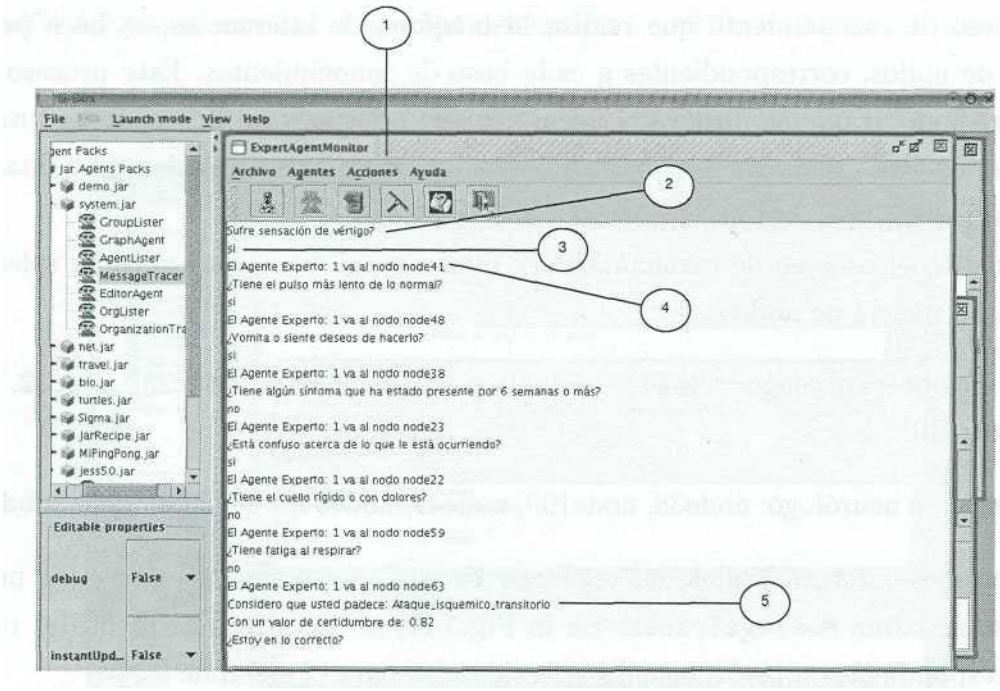

Figura 5.10: *Preguntas y respuestas del Agente Experto neurólogo.*

En la Fig.5.10, se visualiza lo siguiente de acuerdo a las etiquetas:

1. En el área de texto del Panel Principal, se despliegan los mensajes del AE_2, que en el equipo es el médico neurólogo.

2. Las preguntas hechas por el médico neurólogo al usuario, se muestran en lenguaje natural.

3. El usuario introduce la respuesta correspondiente. En este momento, la máquina de inferencias procesa esa respuesta e identifica qué nodo es el siguiente a procesar, enviando esta información al otro AE_i activo, en esta caso es el médico cardiólogo.

4. Después de que se procesa el mensaje internamente, se muestra al usuario, indicando qué nodo será procesado, es decir, qué pregunta será la que despliegue el otro AE_i activo en su interfaz, para que el usuario la conteste.

5. Concluido el proceso de razonamiento, el Agente Experto es capaz de dar su conclusión de acuerdo a las respuestas introducidas por el usuario.

El proceso de razonamiento que realiza la máquina de inferencias, se hace por medio de árboles de nodos, correspondientes a cada base de conocimientos. Este proceso permite visualizar el nodo al que se dirigirá el agente experto después de la respuesta introducida por el usuario. Cada nodo dentro del árbol, representa cada una de las preguntas que se muestran al usuario a través de la interfaz de cada AE_i.

Para ilustrar el proceso de razonamiento e intercambio de mensajes, se ha seleccionado la siguiente secuencia de nodos:

- AE_1: médico cardiólogo: node42, node41, node48, node38, node23, node22, node59, node63, R6.
- AE_2: médico neurólogo: node38, nodel09, node41, node34, node28, node54, node22, R6.

Los mensajes intercambiados, se registran en un contenedor de mensajes propio de MadKit que se llama MessageTracer. En la Fig.5.11, se muestran los mensajes intercambiados correspondientes a las secuencias seleccionadas para el ejemplo.

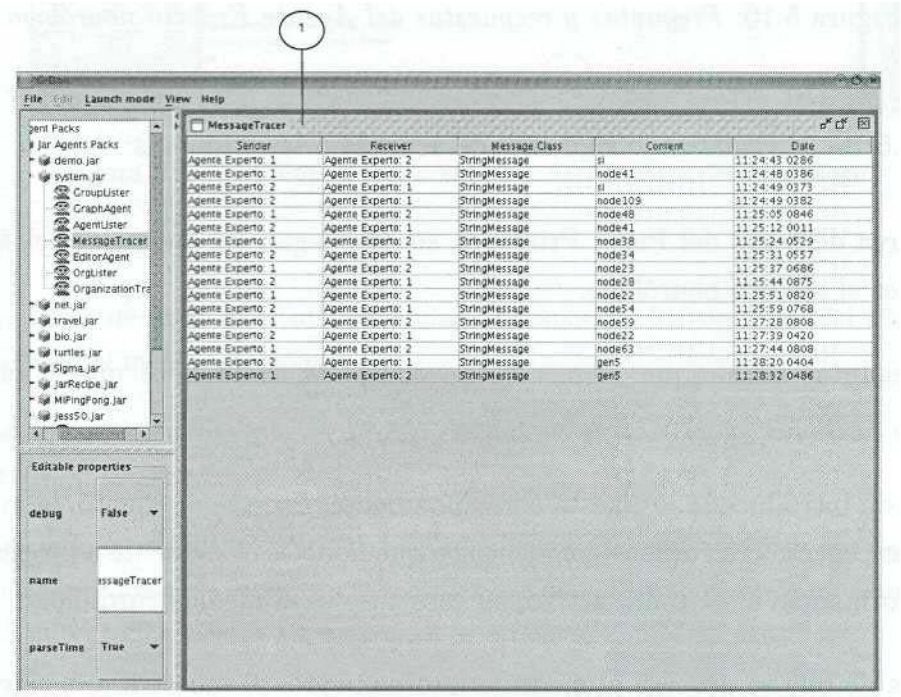

Figura 5.11: *Proceso de razonamiento del árbol de nodos del AgenteExperto 2.*

La *Prueba 2* realizada en el SiCAE, corresponde a un equipo de Agentes Expertos especialista en: enfermedades cardiovasculares y gastrointestinales respectivamente.

En la Fig.5.12 se observa la activación del equipo:

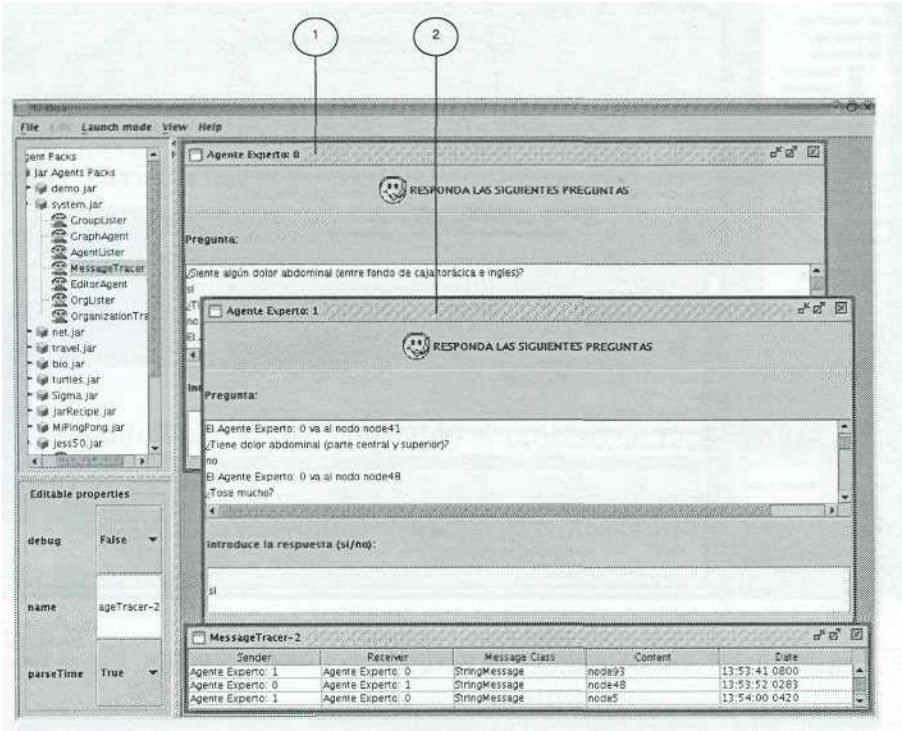

Figura 5.12: *Activación de un equipo de dos agentes expertos, dominio de aplicación de enfermedades.*

1. Activación del AE_0, médico especialista en enfermedades cardiovasculares, con la serie de preguntas mostradas al usuario.
2. Activación del AE_1, médico especialista en enfermedades gastrointestinales, se muestra la serie de preguntas que el usuario deberá responder.

La Fig.5.13, se observan las conclusiones arrojadas por cada AE_i, después del proceso de razonamiento.

1. Conclusión dada por el AE_0
2. Conclusión dada por el AE_1

La secuencia de preguntas, respuestas y mensajes del médico cardiólogo, se detallan en la Fig.5.14.

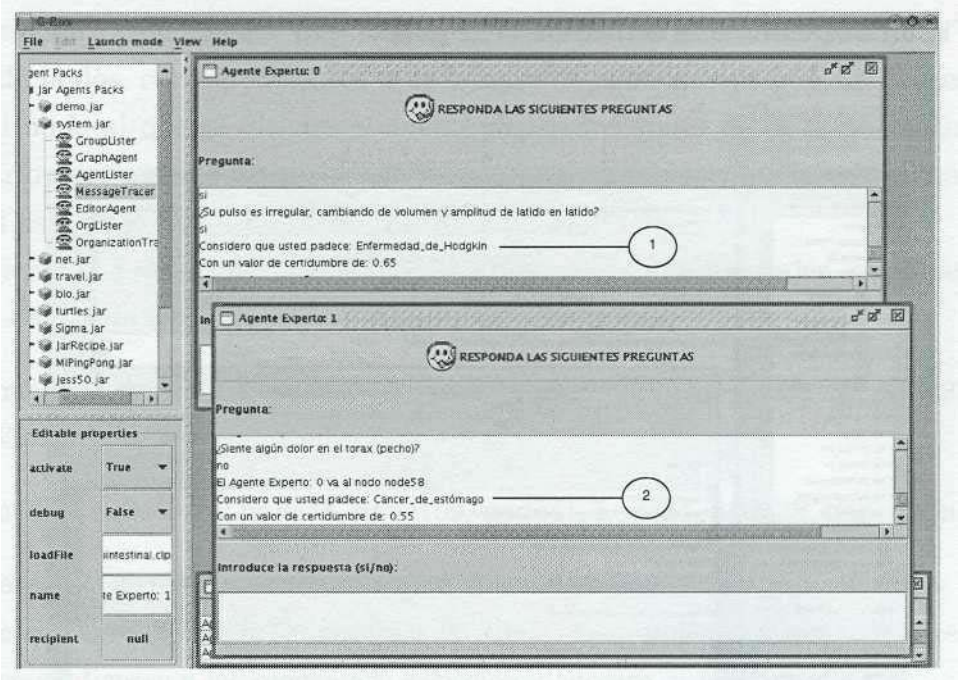

Figura 5.13: *Conclusiones generadas por cada Agente Experto.*

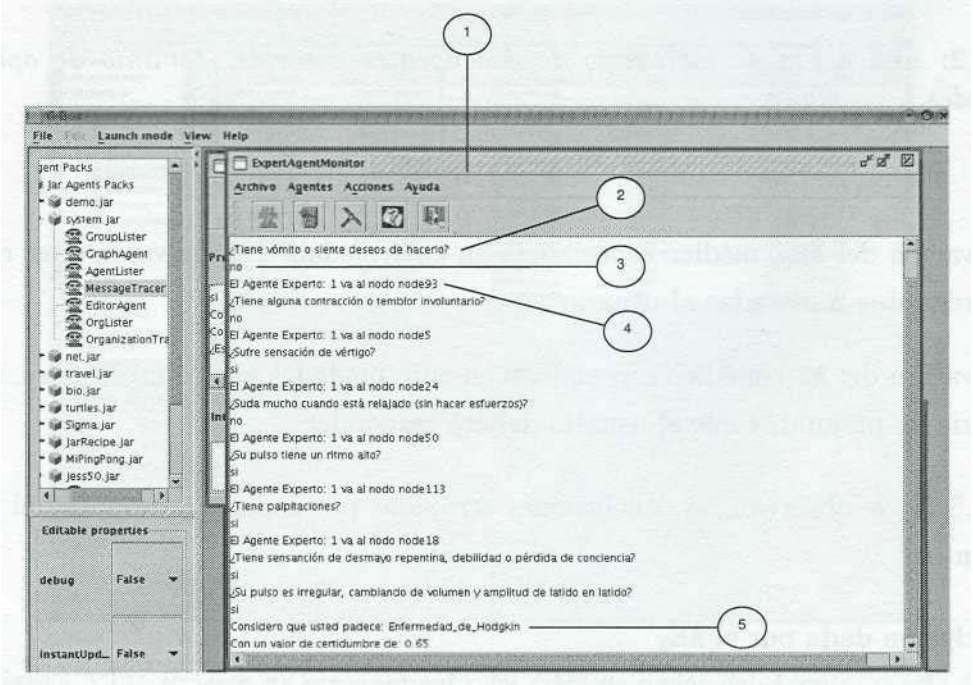

Figura 5.14: *Preguntas y respuestas del Agente Experto cardiólogo.*

1. Área de texto del Panel principal, donde se visualiza la secuencia de mensajes del AE_0.

2. Primer pregunta mostrada al usuario.

3. Respuesta del usuario a la pregunta 1, esta respuesta es procesada por la máquina de inferencias y enviada como mensaje al AE_1, para indicar cuál es el siguiente nodo a procesar.

4. En base a la respuesta introducida por el usuario para la pregunta 1, se muestra cómo es que el AE_1 indica el siguiente nodo a procesar.

5. Respuesta obtenida por el AE_0, concluido el proceso de razonamiento.

Así mismo, la secuencia de preguntas, respuestas y mensajes del médico gastroenterólogo, se ilustran en la Fig.5.15.

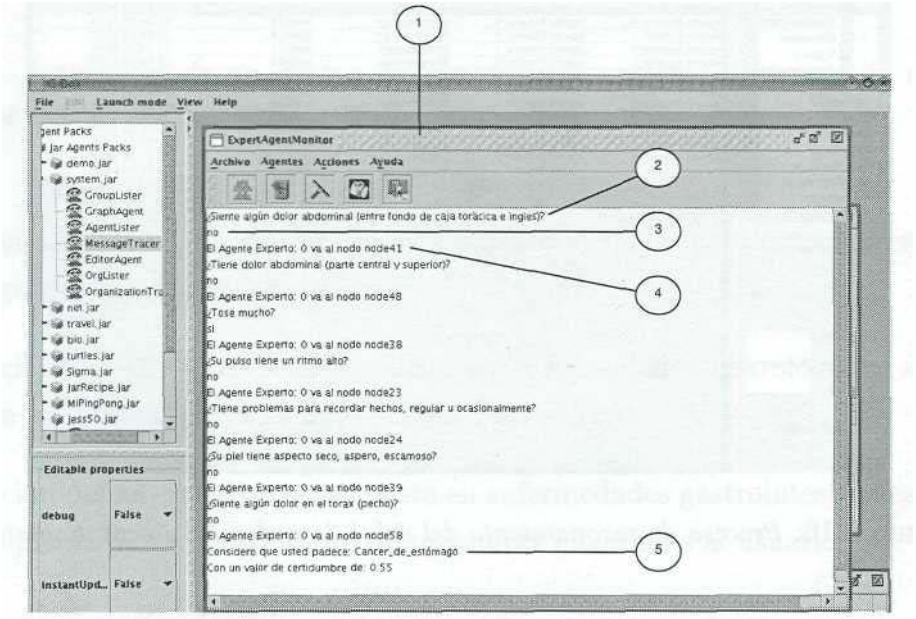

Figura 5.15: *Preguntas y respuestas del Agente Experto gastroenterólogo*

1. Área de texto del Panel principal, donde se visualiza la secuencia de mensajes del AE_1.

2. Primer pregunta mostrada al usuario.

3. Respuesta del usuario a la pregunta 1, misma que es procesada por la máquina de inferencias y enviada como mensaje al AE_0, para dar a conocer cuál es el siguiente nodo a procesar.

4. De acuerdo a la respuesta introducida por el usuario para la pregunta 1, se muestra cómo el AE_0 indica el siguiente nodo a procesar.

5. Respuesta obtenida por el AE_1, después del proceso de razonamiento.

El proceso de cooperación de la *Prueba 2,* se muestra a continuación en la Fig.5.16.

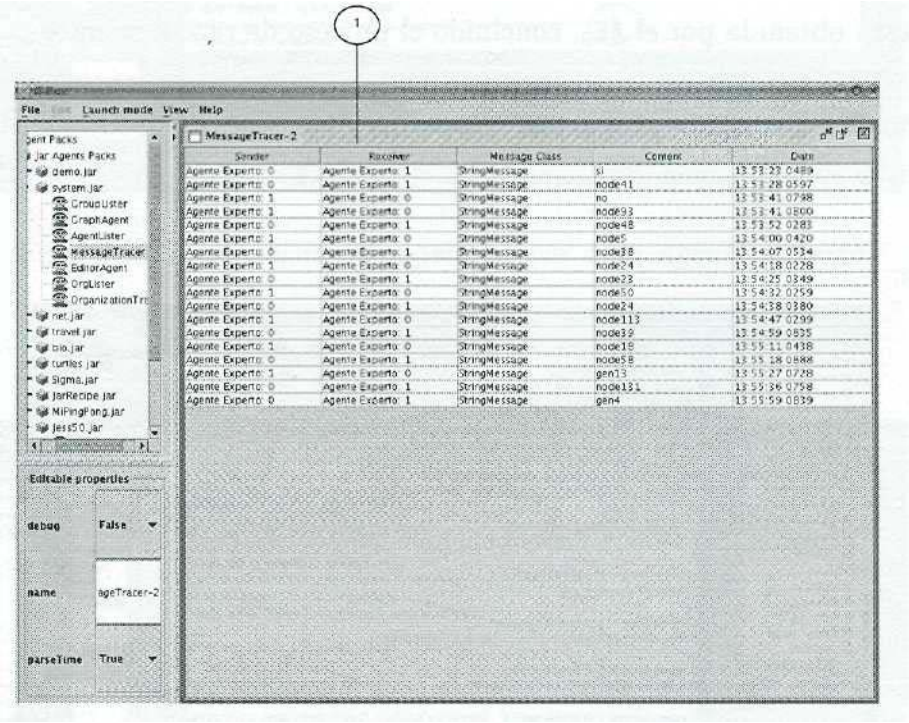

Figura 5.16: *Proceso de razonamiento del árbol de nodos del AgenteExperto 2.*

1. En el contenedor de mensajes MessageTracer del ambiente Gbox, se almacenan todos los mensajes intercambiados entre los AE_i que se activaron para esta segunta prueba.

La *Prueba 3* realizada al SiCAE, corresponde a un equipo de Agentes Expertos especialistas en: enfermedades cardiovasculares, neurológicas y gastrointestinales respectivamente. En la Fig.5.17 se observa la activación del equipo de especialistas:

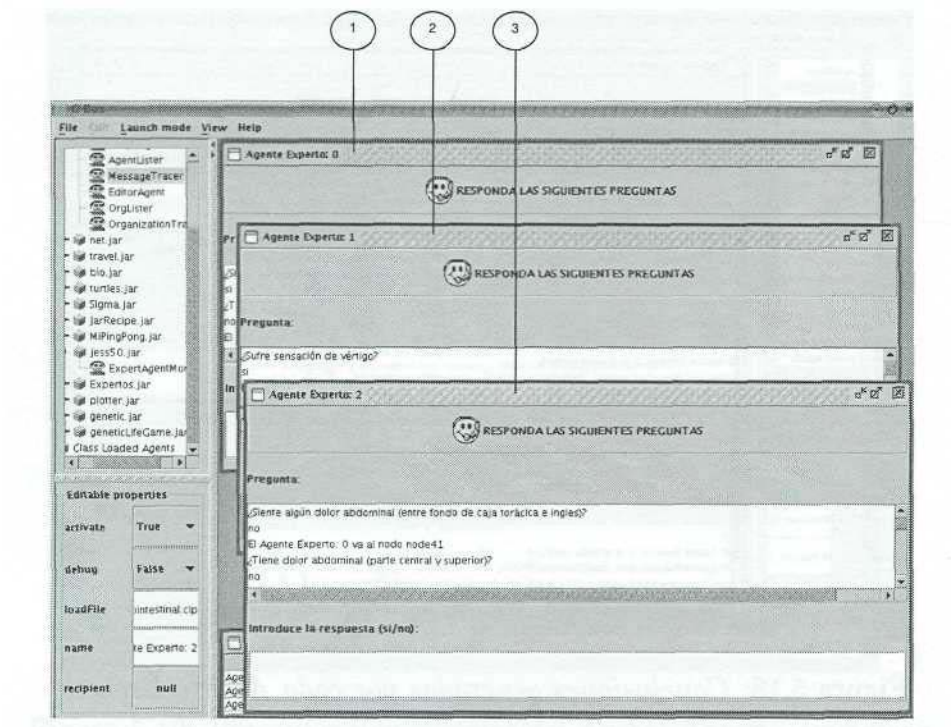

Figura 5.17: *Activación de un equipo de dos agentes expertos, dominio de aplicación de enfermedades.*

1. Activación del AE_0, médico especialista en enfermedades cardiovasculares, con la serie de preguntas mostradas al usuario.

2. Activación del AE_1, médico especialista en enfermedades neurológicas, se muestra la serie de preguntas que el usuario deberá responder.

3. Activación del AE_1, médico especialista en enfermedades gastrointestinales, en su interfaz se aprecia parte del conjunto de preguntas mostradas al usuario.

En la Fig.5.18, se ilustran las conclusiones arrojadas por cada AE_i, una vez que ha finalizado el proceso de razonamiento.

1. Conclusión obtenida por el AE_0.

2. Conclusión obtenida por el AE_1.

3. Conclusión obtenida por el AE_2.

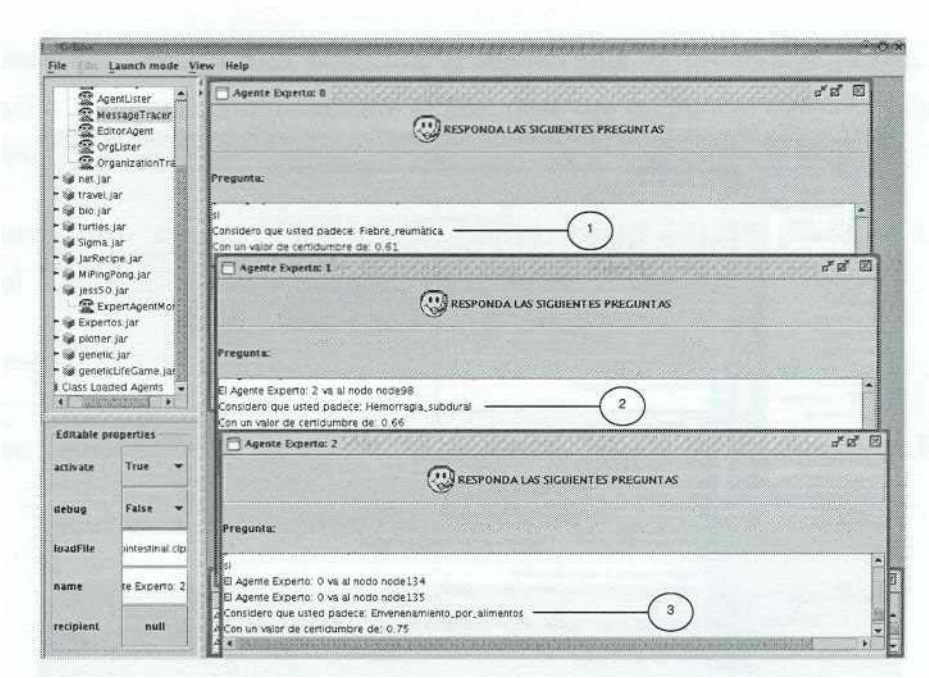

Figura 5.18: *Conclusiones generadas por cada Agente Experto.*

La secuencia de preguntas, respuestas y mensajes del médico cardiólogo, se detallan en la Fig.5.19.

1. Área de texto del Panel principal, donde se visualiza la secuencia de mensajes del AE_0.
2. Primer pregunta mostrada al usuario.
3. Respuesta del usuario a la pregunta 1, esta respuesta es procesada por la máquina de inferencias y enviada como mensaje a los AE_i y AE_2, para indicar cuál es el siguiente nodo a procesar.
4. En base a la respuesta introducida por el usuario para la pregunta 1, se muestra cómo los AE_i y AE_2 indican el siguiente nodo a procesar.
5. Respuesta obtenida por el AE_0, concluido el proceso de razonamiento.

La secuencia de preguntas, respuestas y mensajes del médico neurólogo, se muestran en la Fig.5.20.

1. Área de texto del Panel principal, donde se visualiza la secuencia de mensajes del AE_1.

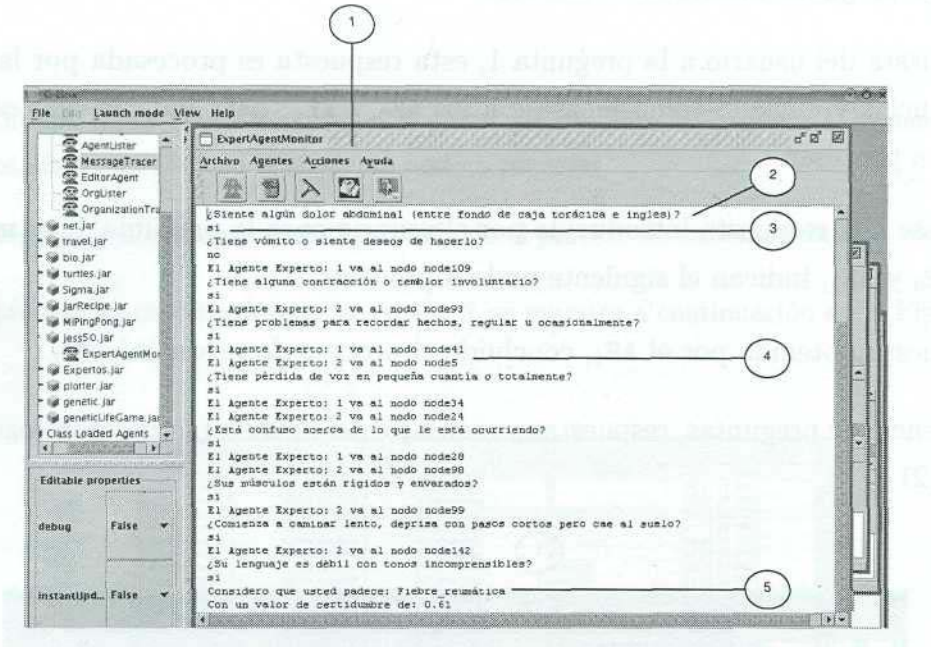

Figura 5.19: *Preguntas y respuestas del Agente Experto cardiólogo.*

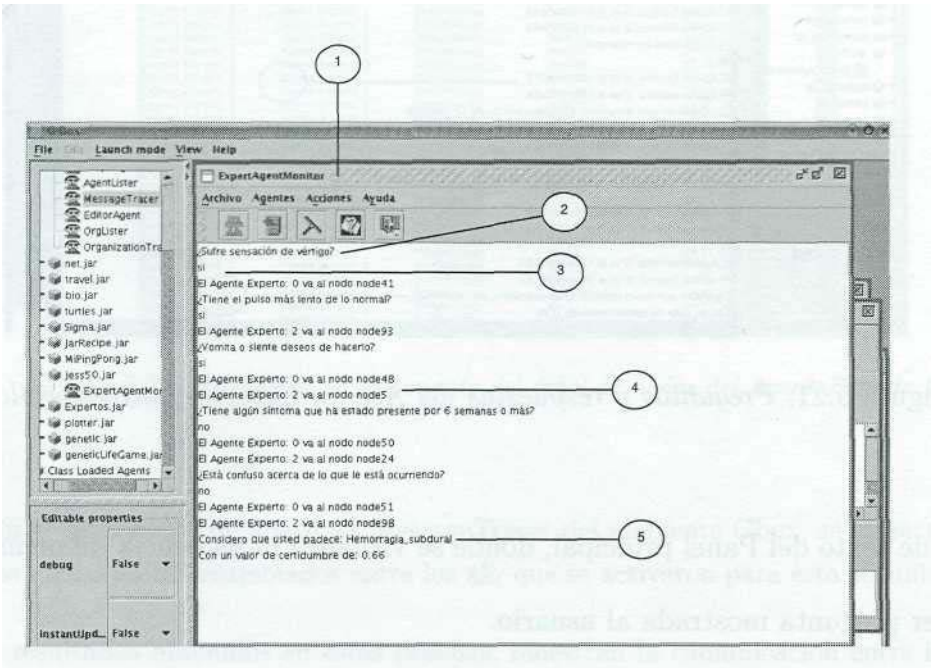

Figura 5.20: *Preguntas y respuestas del Agente Experto neurólogo.*

2. Primer pregunta mostrada al usuario.

3. Respuesta del usuario a la pregunta 1, esta respuesta es procesada por la máquina de inferencias y enviada como mensaje a los AE_0 y AE_2, para indicar cuál es el siguiente nodo a procesar.

4. En base a la respuesta introducida por el usuario para la pregunta 1, se muestra cómo los AE_0 y AE_2, indican el siguiente nodo a procesar.

5. Respuesta obtenida por el AE_1, concluido el proceso de razonamiento.

La secuencia de preguntas, respuestas y mensajes del médico gastroenterólogo, se detallan en la Fig.5.21.

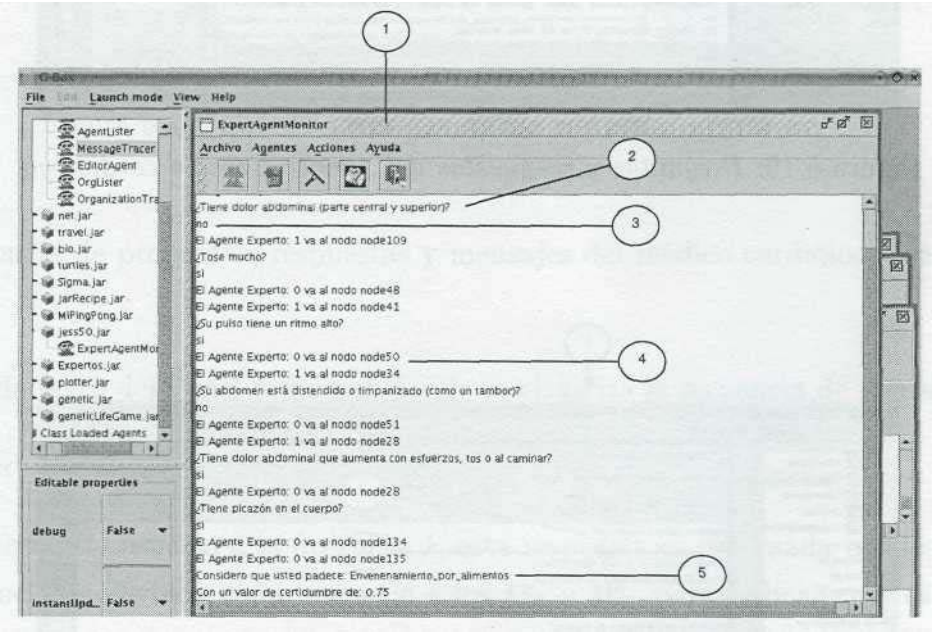

Figura 5.21: *Preguntas y respuestas del Agente Experto gastroenterólogo.*

1. Área de texto del Panel principal, donde se visualiza la secuencia de mensajes del AE_2.

2. Primer pregunta mostrada al usuario.

3. Respuesta del usuario a la pregunta 1, esta respuesta es procesada por la máquina de inferencias y enviada como mensaje a los AEo y $AE_{1;}$ para indicar cuál es el siguiente nodo a procesar.

4. En base a la respuesta introducida por el usuario para la pregunta 1, se muestra cómo los AE_0 y AE_i indican el siguiente nodo a procesar.

5. Respuesta obtenida por el AE_2, concluido el proceso de razonamiento.

El proceso de cooperación de la *Prueba 3,* se muestra a continuación en la Fig.5.22.

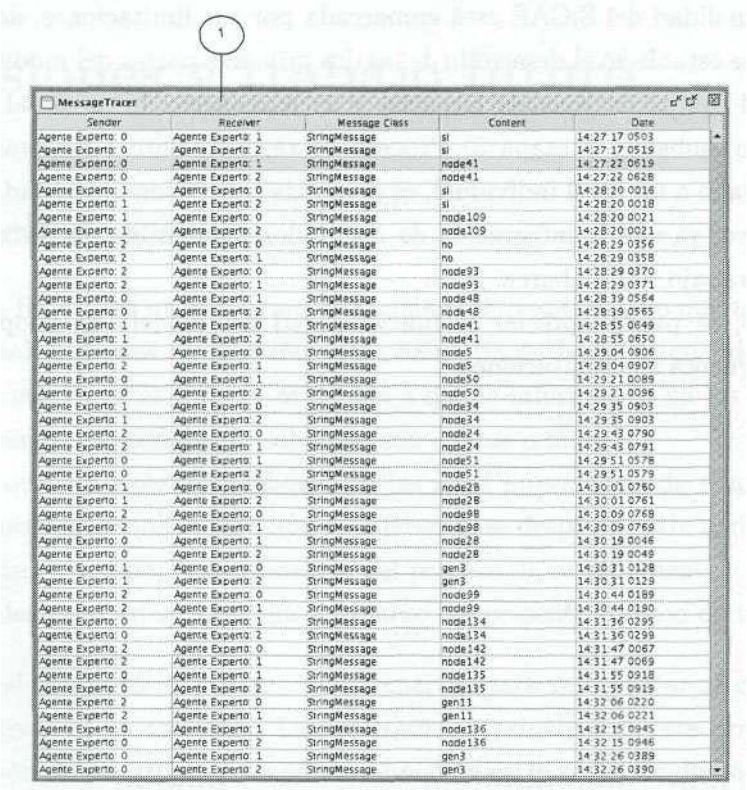

Figura 5.22: *Proceso de razonamiento del árbol de nodos del AgenteExperto 2.*

1. En el contenedor de mensajes MessageTracer del ambiente Gbox, se almacenan todos los mensajes intercambiados entre los AE_i que se activaron para esta segunta prueba.

Los resultados obtenidos en estas pruebas, muestran la comunicación entre los agentes expertos obtenida mediante el modelo de cooperación propuesto en este trabajo. Se logró un intercambio de mensajes de tipo informativo entre los agentes expertos que forman el equipo para solucionar un problema específico.

5.3. Análisis de resultados

Estas pruebas se realizaron de acuerdo al caso de estudio planteado de Agentes Expertos especialistas médicos, obteniendo diversos resultados. En los tres casos se puede observar que los Agentes Expertos activados son capaces de cooperar entre ellos a través del intercambio de mensajes, así como también obtener una conclusión individual.

La funcionalidad del SiCAE está enmarcada por sus limitaciones, debido a que en este trabajo sólo se estableció el desarrollo de las dos primeras partes del modelo: 1) Coordinación y Control del equipo de Agentes Expertos y 2) Asignación de tareas al equipo de Agentes Expertos. Sin embargo la etapa 3) Proceso de razonamiento e integración de resultados, está desarrollado a un nivel individual, es decir, las conclusiones son dadas por cada Agente Experto activo, ya que la integración de resultados requiere de un mecanismo de votación que en este trabajo no se abarca.

Asimismo, se puede apreciar la funcionalidad del modelo de cooperación propuesto, tomando en cuenta sus limitaciones.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

El algoritmo RETE, es un algoritmo de razonamiento optimizado que utiliza la posición actual en el árbol de nodos de decisión como memoria de trabajo: cuando la máquina de inferencia se encuentra en el nodo n, es debido a que la información en los nodos previos es conocida (las premisas anteriores se sabe que son ciertas o falsas).

Para lograr que dos agentes colaboren en las fases intermedias de razonamiento (entre nodos), se requiere lograr que la máquina de inferencia se desplace entre nodos no adyacentes y conserve al mismo tiempo, la consistencia del proceso de razonamiento.

Los principales resultados obtenidos a lo largo del presente trabajo de tesis son:

1. Se aplicó el concepto de sistema multiexpertos para representar al conjunto de individuos activos en un ambiente. La característica principal de este tipo de sistemas, es la emulación de un grupo de expertos humanos en la toma de decisiones. La ventaja que ofrece un sistema multiexpertos, es la modularidad, que soluciona el problema de los SBC de tener una sola base de conocimientos, que puede llegar a ser muy grande y por lo tanto problemática para su mantenimiento.

2. Se complementó la documentación de la estructura de la plataforma Jess. Durante el proceso de investigación sobre esta plataforma, se encontraron deficiencias en cuanto a la disponibilidad de información clara sobre la estructura de Jess, ya que sólo se encontró documentación sobre su manejo. Por ello, fue necesario elaborar el Diagrama de Clases y el cuadro descriptivo de las relaciones de dichas clases. Esta información técnica aportada puede servir para el desarrollo de trabajos futuros que se desprendan de esta tesis.

3. Se desarrolló un esquema de integración de la plataforma MadKit (para el desarrollo de SMA) y Jess (shell para programación de SE), mediante el cual es posible lanzar agentes provistos de un SE, que interactúan dentro de un ambiente multiagentes a través del intercambio de mensajes.

4. Se desarrolló un modelo de sistema de agentes expertos cooperativos, cuya implementación se considera otra aportación de este trabajo, aún con las dificultades encontradas para su logro. Los alcances de este objetivo se limitan a una cooperación entre agentes a un primer nivel, es decir, los mensajes enviados entre los agentes expertos son de tipo informativo. La fase para lograr una integración de tal información intercambiada y obtener una conclusión global, es desarrollar un/mecanismo de mantenimiento del proceso de razonamiento, llevado a cabo por el algoritmo RETE. Este mecanismo no se desarrolla en este trabajo debido a su magnitud y complejidad.

5. Se desarrolló un framework que consta de una biblioteca de clases que genera equipos de agentes expertos. Esta biblioteca de clases se considera de dominio general, ya que las bases de conocimiento de los agentes expertos pueden crearse dependiendo del área de conocimiento del problema a resolver. Para este trabajo de tesis se realizaron pruebas con un equipo de agentes médicos especialistas en Gastroenterología, Neurología y Cardiología.

A lo largo del desarrollo de esta tesis, se encontraron diversos trabajos que de alguna manera se relacionan. Una primera aproximación a la elaboración de múltiples bases de conocimiento para la solución de problemas es presentado en: *Consensus in a Multi-Expert System* [Keung-Chi, 1990], donde la idea principal es tener una naturaleza distribuida de experiencia y formular la agregación de diversas opiniones, pero este trabajo no tiene conexión con agentes. Otro trabajo encontrado es: *MACRON: An Architecture for Multi-agent Cooperative Information Gathering* [Decker et al., 1995], este trabajo plantea una arquitectura de cooperación utilizando fuentes de información en Internet como grupos de noticias, archivos y revistas. A pesar de que este trabajo plantea una arquitectura de cooperación, no cuenta con la integración de algún proceso de razonamiento para el intercambio de información.

También hay otros trabajos que hablan de la cooperación en Sistemas Multiagentes (SMA) como: *On Cooperation in Multi-Agerú Systems* [Doran, Franklin y Jennings, 1997], *A framework for argumentation-based negotiation* [Sierra et al., 1998], pero no integran procesos de razonamiento, ni bases de conocimiento.

Otra aproximación encontrada es: *A Simulation Approach Based on Negotiation and Cooperation Between Agents: A Case Study* [Fischer y Chaib-draa, 1999]. Aquí se integran agentes y bases de conocimiento, enfocados a resolver problemas de transporte, es decir, no es una aplicación de dominio general.

En este trabajo de tesis propuesto, se integran Sistemas Multiagentes y Sistemas Expertos, llamados Agentes Expertos, dedicados a resolver problemas en algún dominio de aplicación. Para lograr esto se integraron dos plataformas de desarrollo: MadKit y Jess. Además se propone un modelo de cooperación para lograr la comunicación entre estos agentes que tienen una base de conocimiento. A pesar de sus limitaciones, se obtuvo una comunicación a nivel informativo entre los agentes expertos activos, lo que da la pauta para poder desarrollar otros trabajos que permitan completar este proyecto.

Para la solución de problemas complejos en diversas áreas, se sugiere utilizar una combinación de diversas técnicas de la Inteligencia Artificial para obtener mejores resultados, que los que se lograrían utilizando una sola técnica para lograr la solución. A esta combinación de técnicas para la solución de problemas se le llama Clasificación Colectiva y se pueden incluir: Redes Neurnales Artificiales, Lógica Difusa, Razonamiento Basado en Casos, Algoritmos Genéticos y Sistemas Multiagentes.

6.2. Trabajo futuro

Existe mucho trabajo aún por realizar en dirección a esta tesis, misma que puede servir de base para la continuación de diversos trabajos. Entre estos trabajos se pueden citar los siguientes:

1. Desarrollar un mecanismo de mantenimiento del proceso de razonamiento entre los agentes expertos, que permita un mayor grado de cooperación entre los agentes.

2. Desarrollar un mecanismo de votación, que permita obtener una conclusión global por medio de un agente arbitro, en base a las opiniones individuales, con pesos de certidumbre entre los agentes expertos activos. Se sugiere la utilización de Clasificación Colectiva para obtener una mejor solución.

3. Implementar el modelo propuesto en este trabajo en un ambiente distribuido, dando la oportunidad de tener varios usuarios interactuando con diversos grupos de agentes expertos.

4. Desarrollar e implementar un modelo que permita intercambiar información con expertos humanos en línea, dentro de un sistema distribuido.

5. Implementar un caso de estudio real, en el que se pueda observar la cooperación a nivel organizacional, realizando diversas pruebas para determinar la funcionalidad del modelo propuesto en este trabajo.

6. Se propone desarrollar un sistema multiagentes expertos, que permita la interacción distribuida entre dos equipos, uno que se ejecute en un sistema PDA y otro en una computadora portátil, la comunicación entre ambos equipos es por medio de puertos infrarojos. Actualmente este proyecto se está iniciando como tesis de licenciatura.

Así mismo, en el proceso de realización de este trabajo de tesis se sometió un artículo a un congreso internacional, obteniendo a la fecha los siguientes reconocimientos:

- Diploma otorgado por la presentación de la conferencia: *"Cooperación en equipos de agentes expertos",* en el XXIV Congreso Internacional de Ingeniería Electrónica, ELECTRO 2002. Llevado a cabo del 21 al 25 de octubre de 2002, en el Tecnológico de Chihuahua.

- Memoria en extenso de la ponencia: *"Cooperación en equipos de agentes expertos". "ELECTRO 2002, XXIV Congreso Internacional de Ingeniería Electrónica".* Volumen XXIV, ISSN: 1405-2172. Instituto Tecnológico de Chihuahua, División de Estudios de Posgrado e Investigación, 21-25 Octubre 2002, págs. 333-338.

- Invitación a participar en el 5o. Simposium Estatal de Informática del Centro de Cómputo Académico de la UAEH, Abril 2003. Evento que se lleva a cabo cada año para celebrar el aniversario de dicho centro.

- Reconocimiento entregado por la participación en el 5o. Simposium Estatal de Informática del Centro de Cómputo Académico de la UAEH, con la conferencia: *"Cooperación en equipos de agentes expertos",* llevado a cabo el 30 Abril de 2003.

- Publicación del artículo: *"Cooperación en equipos de agentes expertos",* en el libro titulado: *Temas Selectos de Computación 2001-2002,* editado por la Universidad Autónoma del Estado de Hidalgo, 2003. ISBN: 9686340-920, págs. 258-266.

Apéndice A

Sistemas Basados en Conocimiento

A.l. Sistemas Expertos

La tecnología de los *SE* deriva de la Inteligencia Artificial (IA): una rama de las ciencias computacionales interesada en el diseño e implementación de programas, que sean capaces de emular las habilidades cognitivas humanas como: la solución de problemas, la percepción visual y el entendimiento del lenguaje. Esta tecnología ha sido satisfactoriamente aplicada en diversos dominios [Jackson, 1999].

Un *Sistemas Experto (SE)* es un programa de computadora que representa y emula el razonamiento basado en el conocimiento de un especialista, con la finalidad de resolver problemas o dar consejos en un dominio específico. Un *SE* puede destinarse a cumplir totalmente una función que normalmente requiere la especialización humana, o puede jugar el papel de asistente en la toma de decisiones humanas [Jackson, 1999].

A.2. Elementos de un Sistema Experto

Todo sistema experto tiene una estructura definida, en la Fig. A.l se muestran los elementos que la conforman [Giarratano y Riley, 2001].
A continuación se explica cada uno de los componentes de los SE.

A.2.1. Base de Conocimientos

Contiene conocimiento especializado que es extraído de un experto en el dominio. Existen varias técnicas de representación del conocimiento entre las que se encuentran las *reglas, redes semánticas, marcos, guiones, lenguajes de representación del conocimiento* como KL-1

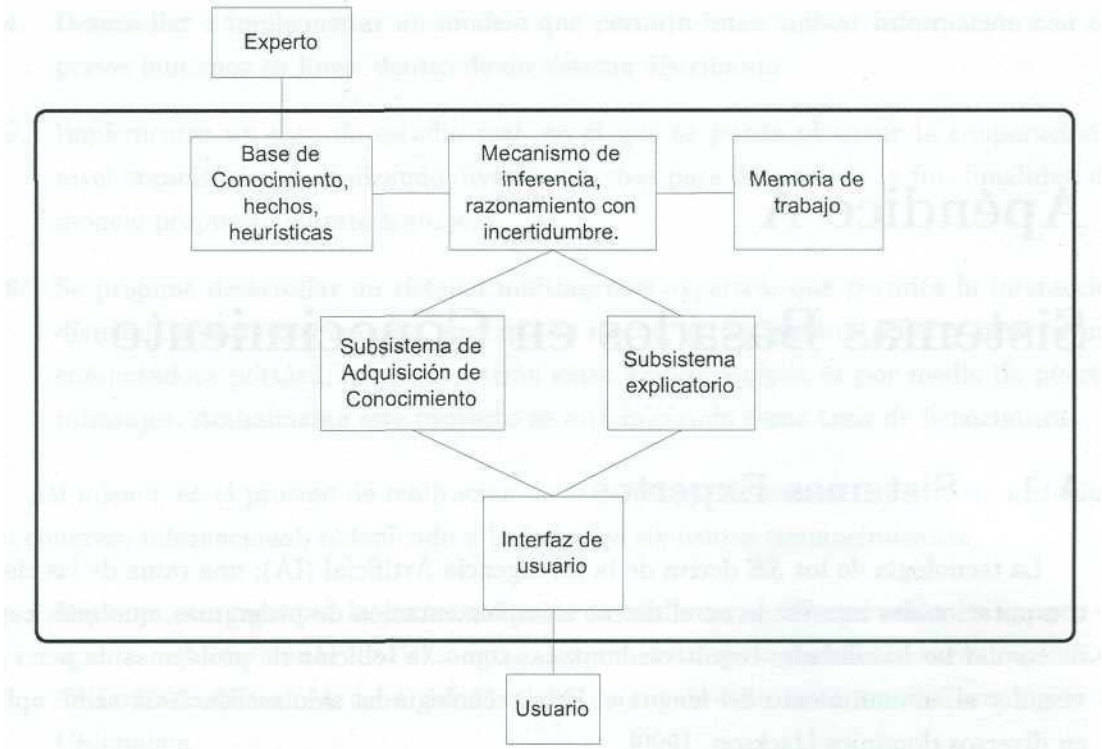

Figura A.l: *Componentes de un Sistema Experto*

y KRYPTON, *gráficas conceptuales,* entre otros.

Uno de los tipos más comunes y populares de SE es aquel basado en reglas. Los **sistemas de producción** se utilizaron inicialmente en lógica simbólica por Post, dando como resultado que cualquier sistema de matemáticas o lógica se podía escribir como cierto tipo de reglas de producción. En este paradigma de programación, las reglas de producción son utilizadas como representaciones heurísticas, o *"reglas empíricas",* las cuales especifican un conjunto de acciones a realizar en una situación dada [Jackson, 1999].

Las características de las reglas son: *naturaleza modular,* que facilita la encapsulación del conocimiento y permiten que el SE se pueda expandir; medios de explicación, presentan la cadena de razonamiento que condujo a cierta conclusión; semejanza con el proceso cognitivo humano, la representación del tipo SI condición... ENTONCES acción facilita la explicación, de los especialistas sobre la estructura del conocimiento que se trata de obtener de ellos, además de que son un modelo natural de la forma en que los humanos resuelven problemas.

A pesar de la utilidad de las reglas de producción, se encontró una limitante para la

programación que es la falta de una estrategia de control que guíe la aplicación de las reglas. Markov evoluciona las reglas de producción especificando una estructura de control para los sistemas de producción, surgiendo así los **algoritmos de Markov.** Un algoritmo de Markov es un grupo ordenado de producciones que se aplican en orden de prioridad a un ahilera de entrada. Si la regla de prioridad más alta no es aplicable, entonces aplica la siguiente y así sucesivamente.

A pesar de que el algoritmo de Markov se puede utilizar como la base de un SE, resulta poco eficiente en sistemas con muchas reglas. Es necesario que un algoritmo conozca las reglas y pueda aplicar cualquiera de ellas sin que tenga que probar cada una en forma secuencial. Una solución a este problema es el algoritmo RETE aplicado en el shell del SE OPS (Oficial Production System). El algoritmo RETE es un rápido igualador de patrones que sólo busca los cambios en las correspondencias de cada ciclo, lo que le da velocidad. Se han desarrollado diferentes versiones de lenguajes OPS y su shell, incluyendo OPS2, OPS4, OPS5 y OPS83. Estos shells no son los únicos que utilizan el algoritmo RETE para el manejo de las reglas de producción, también está CLIPS y Jess. Debido a que Jess es el shell a utilizar en este trabajo, se explica ampliamente en el Capítulo 3.

A.2.2. Máquina de Inferencia

Se refiere a los mecanismos de razonamiento para manipular el conocimiento o información simbólica que está en la Base de Conocimientos. Decide cuáles reglas satisfacen los hechos u objetos, da prioridad a las reglas satisfechas y ejecuta la regla con la prioridad más elevada.

Existen diversos métodos de inferencia como son: *deducción, inducción, intuición, heurística, ensayo y error, abducción, predeterminación, autoepistemología, no monótono* y *analogía.*

Uno de los métodos más comunes para formular inferencias es la lógica deductiva, que se ha utilizado para determinar la validez de un argumento. En un argumento se tienen premisas o antecedentes y conclusiones o consecuencias. La característica esencial de la lógica deductiva es que la conclusión verdadera se obtiene de premisas verdaderas.

Dos métodos de inferencia que se utilizan en los SE son: *encadenamiento hacia delante* y *encadenamiento hacia atrás,* aunque se pueden utilizar otros métodos para necesidades más específicas. El *encadenamiento hacia delante* es el razonamiento desde los hechos hacia las conclusiones que resultan de ellos. El *encadenamiento hacia atrás* implica el razonamiento en reversa desde una hipótesis, por lo que se tiene que comprobar una posible conclusión a los hechos que la sustentan. Por ejemplo OPS5 y CLIPS están diseñados para el encadenamiento

hacia delante, mientras que EMYCIN ejecuta encadenamiento hacia atrás, y otros tipos de mecanismo de inferencia como ART y KEE, ofrecen ambos.

El método de inferencia de ensayo y error se utilizó en el primer SE llamado DENDRAL, creado para ayudar a identificar estructuras moleculares orgánicas. Una variación es el méto - do llamado plan-ensayo y error que se utilizó en el SE de diagnóstico médico MYCIN.

El programa de adquisición de conocimiento TEIRESIAS para MYCIN utiliza *meta-conocimiento* que puede ser de dos tipos: una estrategia de control que indica la forma en que se aplicarán las reglas y el tipo de modelo de regla que determina si la nueva regla está en una forma correcta.

Los métodos de razonamiento pueden utilizar *razonamiento exacto* para determinar cuál es la mejor conclusión, pero otras aplicaciones requieren *razonamiento inexacto* o *incertidumbre* que tenga relación con hechos y reglas inciertos. Algunos ejemplos de SE que manejan la incertidumbre son MYCIN (diagnóstico médico) y PROSPECTOR (explotación mineral), para tratar la incertidumbre en estos SE se utilizan los *factores de certidumbre.* Posteriormente surgen nuevas teorías para la representación de la incertidumbre como la *lógica difusa* y *la teoría de Dempster Shafer,* también resurge el uso de la probabilidad para el manejo de la incertidumbre con el desarrollo de las *redes Bayesianas.*

A.2.3. Memoria de trabajo

Es el lugar donde se almacenan los datos de entrada y conclusiones intermedias que se generan durante el proceso de razonamiento.

A.2.4. Subsistema de Adquisición de Conocimiento

Es un susbistema que tiene como objetivo auxiliar al experto en la construcción de las bases de conocimiento. Esta ayuda consiste en la colección del conocimiento necesario para la solución de un problema, así como la elaboración de la base de conocimiento, lo cual es considerado como un "cuello de botella" en la construcción de Sistemas Expertos.

No obstante las ventajas y dominios de aplicación de los *SE,* existen algunas desventajas, particularmente relacionadas con la *Adquisición del Conocimiento.*

La *Adquisición del Conocimiento* es la transferencia y transformación a un programa, de la especialización o experiencia para solucionar un problema. Esta transferencia de información tiene que llevarse a cabo mediante largas e intensas entrevistas entre el ingeniero del conocimiento y el experto en el dominio. Este procedimiento puede producir al día de dos a

cinco reglas heurísticas, lo cual ha llevado a algunos investigadores a enfocarse en este cuello de botella de adquisición de conocimiento de las aplicaciones de *SE,* ya que la productividad de éstos sistemas puede resultar pobre [Jackson, 1999].

Algunos puntos a destacar sobre la baja productividad de las reglas para los SE son:

1. Los diferentes especialistas utilizan su propia jerga, y resulta difícil encontrar un lenguaje común.

   Por ejemplo, un estratega militar puede hablar de la "postura agresiva" de un poder exterior, sin ser capaz de especificar exactamente lo que distingue tal postura de una no amenazante.

2. Los hechos y principios son la base de muchos dominios de interés que no pueden ser caracterizados en términos de teorías matemáticas o modelos determinísticos en los que sus propiedades son bien entendidas.

   Por ejemplo, un experto en finanzas puede saber que ciertos eventos causan que las acciones del mercado suban o bajen, pero el mecanismo y magnitud exactos que intervienen en esos efectos no pueden ser predecidos o identificados con certeza.

3. Los expertos necesitan saber más que solo los hechos o principios de un dominio para resolver un problema. Por ejemplo, generalmente saben que tipo de información es relevante para un determinado juicio, que tan confiables son las fuentes de información y cómo hacer que un problema difícil se convierta en fácil dividiéndolo en subproblemas, los cuales podrían resolverse independientemente.

   Éste tipo de conocimiento es más difícil de sustraer, ya que está basado en la experiencia personal más que en un entrenamiento formal.

4. La experiencia humana, aún en un dominio relativamente reducido, es con frecuencia establecido en un contexto amplio que involucra una cantidad considerable de conocimiento de sentido común relacionado con cada día.

   Como ejemplo se pueden considerar a unos expertos en leyes involucrados en una litigación. Es difícil delinear la cantidad y naturaleza del conocimiento general que es necesario para tratar con un caso arbitrario.

5. Una limitación de los sistemas basados en reglas, es la incapacidad de resolución de conflictos de representación de conocimiento.

En muchos casos se cuenta con varios expertos en un mismo dominio, que pueden tener ideas diferentes e incluso contradictorias para la solución de un problema en particular. Antes de poder codificar este conocimiento, es necesario resolver tales diferencias.

6. Algunos puntos anteriores pueden causar otro problema como la pérdida de información, ocasionando la existencia de datos incompletos para el procesamiento.

7. Cuando se ha completado la adquisición de conocimiento, se puede proceder a la codificación del mismo para que posteriormente pueda ser procesado por la máquina de inferencias. Pero en determinado momento el experto puede decidir que el conocimiento que proporcionó no es suficiente para solucionar el problema, así que es necesario integrar nuevo conocimiento. Este proceso puede ser largo y tedioso, debido a que ese conocimiento puede depender de otro ya existente, siendo necesario reordenarlo para su correcta integración.

8. Otra situación que puede presentarse en la adquisición de conocimiento es que el dominio de éste sea demasiado grande, ocasionando un proceso muy largo para adquirirlo, pérdida de datos importantes y no poder asociarlo correctamente. I

9. Al igual que los expertos, pueden existir varios ingenieros del conocimiento, lo que podría ocasionar un desacuerdo en cuanto a la adquisición de conocimiento, codificación y asociación de datos relevantes para una buena solución.

A pesar de que los *SE* son una herramienta importante en la toma de decisiones, los resultados que proporcionan algunas veces tienen que ser reforzados por el experto humano. Se requieren mecanismos que integren el conocimiento del usuario (su opinión) con la de los SE.

A.2.5. Subsistema Explicatorio

Este subsistema consiste en la explicación de las acciones que lleva a cabo el sistema experto. Esta explicación puede consistir en decir cómo las soluciones finales o intermedias fueron generadas, e inclusive dar la justificación de porqué pueden ser utilizadas esas acciones como datos adicionales al sistema.

A.2.6. Interfaz de Usuario

Este componente de los SE es la forma en la que el sistema se presenta al usuario para su interacción [Criado, 2000].

Al desarrollar un SE y su interfaz, se deben tener en cuenta los siguientes puntos - [Criado, 2000]:

1. **El aprendizaje del manejo debe ser rápido.** El usuario no debe dedicar mucho tiempo al menajo del sistema, es decir, debe ser fácil en su manejo e intuitivo. Debe ser cómodo y relativamente sencillo ya que simula el comportamiento de un experto humano.

2. **Debe evitarse la entrada de datos errónea.** Para que el SE pueda dar un diagnóstico adecuado, debemos darle datos correctos. Por ejemplo, cuando acudimos al médico debemos darle las características correctas de los síntomas, de lo contrario el médico nos dará un diagnóstico equivocado.

3. **Los resultados deben presentarse en forma clara.** El usuario debe comprender el resultado que proporciona el sistema experto, que deberán ser claros y concisos.

4. **Las preguntas y explicaciones deben ser comprensibles.**

Las interfaces de usuario pueden tomar varias formas, las cuales podrán ofrecer al usuario ciertas características dependiendo del tipo de sistema y usuarios.

Interfaces de usuario tradicionales para SE

- **Interfaces basadas en texto:** Esta es la forma más usual de las interfaces de usuario y es muy común en los Sistemas Basados en Conocimiento, ya que permite tener un escenario de entrada/salida durante cualquier operación de pregunta/respuesta [Winstanley, 1987].

  Además puede ser una facilidad utilizar una salida basada en texto que proporcione una información explicatoria. Esto significa que los sistemas con una interfaz basada en texto extraen palabras clave de la estructura de reglas y sus conclusiones, para así poder construir una frase que logre explicar los resultados de una forma significativa.

  Una extensión importante son las interfaces en leguaje natural, capaces de "entender" una entrada escrita en lenguaje común (como el Inglés o Español). Este tipo de entradas

al sistema presenta inconvenientes como la variabilidad que se puede producir en el proceso de análisis del texto.

- **Interfaces gráficas:** Los usuarios de sistemas y en general de un sistema computacional, tienen una marcada afinidad por las imágenes, e inclusive acuden a la representación de algún problemas en forma gráfica, en su proceso de solución [Winstanley, 1987].

  Por ejemplo, en Matemáticas, las gráficas y los diagramas son utilizados constantemente para ayudar a comprender la complejidad de un tema. En el diseño de programas basados en conocimiento esta aproximación es relevante ya que la complejidad se presenta en las asociaciones presentes dentro de una base de conocimientos muy grande.

- **Interfaces de red:** La Internet permite a los desarrolladores proporcionar servidores de conocimiento inteligentes. Tener SE corriendo en servidores pueden dar soporte a grandes grupos de usuarios que se comunican con el sistema a través de la red. En este sentido, las interfaces de usuario basadas en protocolos de red, proporcionan acceso a los servidores de conocimiento. El extenso uso de la Internet da la oportunidad de hacer a los sistemas expertos ampliamente disponibles, ya que los usuarios pueden realizar sus tareas de forma remota y los desarrolladores pueden publicar su experiencia a través de la red [Eriksson, 1999].

Los problemas a resolver por los sistemas expertos se pueden clasificar en diversas categorías, dependiento del tipo de problema a tratar. Los sistemas expertos se han aplicado en muchas áreas, por ejemplo: interpretación, predicción, diagnóstico, diseño, planificación, monitorización o supervisión, depuración, reparación, instrucción, control, enseñanza, entre otras [Samper, 2002].

Un *Sistema Experto* puede distinguirse de programas de aplicaciones convencionales en que: [Jackson, 1999]

- Simulan el razonamiento humano acerca del dominio de un problema, en lugar del dominio mismo. Esto distingue a los sistemas expertos de programas más familiares que involucran modelación matemática o simulación por computadora. Esto no quiere decir que el programa es un modelo psicológico fiel del experto, sólo que el enfoque está en la emulación de las habilidades del experto para resolver un problema, esto es, realizar una porción de las tareas relevantes, así como lo hace, o mejor de cómo lo hace el experto.

- Realiza razonamiento sobre las representaciones del conocimiento humano, además de realizar cálculos numéricos o recuperación de datos. El conocimiento en el programa normalmente está expresado en algún lenguaje de propósito general y almacenado de forma separada del código que realiza el razonamiento. Estos distintos módulos del programa son llamados base de conocimiento y máquina de inferencia, respectivamente.

- Resuelve problemas por métodos heurísticos o de aproximación que, a diferencia de las soluciones algorítmicas, no garantizan su éxito. Un método heurístico es esencial mente una regla empírica que tiene codificado el conocimiento necesario para resolver problemas en algún dominio. Los métodos son aproximados en el sentido de que (i) no requieren datos perfectos y (ii) la solución generada por el sistema puede ser propuesta con varios grados de confianza.

A.3. Sistemas Multi-Expertos

Una propuesta para solucionar este problema de adquisición de conocimiento, es tener el conocimiento en módulos, lo cual corresponde a tener varios expertos de uno o varios dominios de forma independiente, de tal forma que se pueda establecer una discusión entre ellos, buscando el consenso para una mejor solución a los problemas.

En la vida real, los grupos de decisiones se caracterizan por tener cooperación, honestidad, experiencia y habilidad. Los miembros de estos grupos comparten un objetivo común y para llegar a él comparten sus opiniones y las integran junto con sus experiencias [Keung-Chi, 1990].

La mayoría de los *SE* tienen una sola base de reglas, así que cualquier extensión a este modelo a grupos de toma de decisiones, debe contemplar la posibilidad de conflictos con reglas codificadas por miembros de diferentes grupos. Ante ésta situación comenzaron a aparecer los sistemas multi-expertos basados en reglas como sistemas consultantes, los cuales hacen recomendaciones de solución basándose en las entradas proporcionadas por múltiples expertos, teniendo así, otra alternativa de representación de conocimiento [Keung-Chi, 1990].

A.3.1. Lenguajes de programación para Sistemas Expertos

Un lenguaje para sistemas expertos es un traductor de comandos con una sintaxis específica, que proporciona un mecanismo de inferencia para ejecutar las instrucciones del lenguaje [Giarratano y Riley, 2001].

Históricamente, los primeros *SBC* fueron desarrollados en lenguajes de programación como LISP y PROLOG.

LISt Processor (LISP) fue el primer lenguaje para procesamiento simbólico, desarrollado por John MacCarthy en 1958 en el Instituto de Tecnología de Massachusetts (MIT). Actualmente LISP se utiliza para la escritura de compiladores, sistemas para diseño VLSE, sistemas para diseño mecánico asistido por computadora, animaciones gráficas y sistemas basados en conocimiento.

PROgramming in LOGic (PROLOG) fue desarrollado en Francia en 1973 por Alain Colmenauer en la Universidad de Marseilles. Su aplicación inicial fue en el procesamiento de lenguaje natural y posteriormente para aplicaciones de IA por su capacidad de manipulación simbólica. A partir de 1981 tuvo una gran difusión a nivel mundial debido a que los japoneses lo comanzaron a utilizar para desarrollar sistemas de cómputo de quinta generación. Actualmente se sigue utilizando y existen varios dialectos para diversas plataformas.

Otro lenguaje utilizado en la ingeniería del conocimiento es Official Production System 5 (OPS5), es un miembro de la familia de lenguajes de programación desarrollados en la Universidad de Carnegie - Mellon. Utiliza la representación del conocimiento en forma de reglas y tiene un mecanismo de encadenamiento hacia adelante como intérprete. Con éste lenguaje se han desarrollado implementaciones comerciales para diferentes plataformas.

CLIPS utiliza una sintaxis e intérprete similares a los de LISP, y la relación entre ellos es que ambos lenguajes hacen uso de la lógica de primer orden para la representación del conocimiento. CLIPS también es un lenguaje de programación con paradigmas múltiples que proporciona soporte para la programación basada en reglas, orientada a objetos y por procedimientos. Las capacidades de inferencia y representación que proporciona este lenguaje son similares a las de OPS5, pero ofrece más funciones. Aunque las reglas que soporta sólo realizan encadenamiento hacia adelante y no hacia atrás. Las opciones de programación orientada a objetos de CLIPS son una combinación híbrida de funciones de otros lenguajes orientados a objetos como CommonLISP y SmallTalk [Giarratano y Riley, 2001].

A medida que el desarrollo de *SBC* iba aumentando en cantidad y complejidad, la comunidad científica comenzó a buscar formas de desarrollar los sistemas en menor tiempo y con menor esfuerzo. Esto dio lugar al surgimiento de los *Shells,* los cuales ofrecen toda la arquitectura de un *SBC.*

Posterior a ese surgimiento, ingresaron al mercado otras herramientas, que además de las opciones de representación del conocimiento, incorporan esquemas de inferencia y control, a éstas herramientas se les llamó *Entornos de Desarrollo de SBC.*

A.3.2. Entornos de desarrollo de Sistemas Basados en Conocimiento

Son herramientas con propósitos especiales, que proporcionan un ambiente de desarrollo de software con componentes básicos para la generación de SE. A través de ellos el usuario sólo debe proporcionar la base de conocimientos de acuerdo a sus requerimientos [Giarratano y Riley, 2001].

Algunos ejemplos de éstos sistemas son:

- ***Shells o Sistemas vacíos:*** EMYCIN, Crystal, Leonardo, XiPlus, EXSYS, VP-Expert, Intelligence Compiler.
- ***Entornos híbridos de desarrollo:*** CLIPS, KEE, ART, EGERIA, Kappa, Nexpert Object, Goldworks, LOOPS, Flavors.

Posteriormente surgieron nuevas versiones y otros shells como lo es Jess (Java Expert System Shell). Derivado a partir de CLIPS, se considera como un Shell o Ambiente de Desarrollo para Sistemas Expertos, ya que tiene su propio lenguaje para el desarrollo de *SE* el cual puede unirse con el lenguaje Java.

El término de *sistema basado en conocimiento,* se utiliza para la aplicación de tecnología basada en el conocimiento, que puede usarse para la creación de sistemas expertos o basados en el conocimiento. Sin embargo, tal y como ocurre con el término inteligencia artificial, es común usar el término sistemas expertos cuando se refieren a ambos tipos de sistemas [Giarratano y Riley, 2001].

A.3.3. Agentes inteligentes de interfaz

Las interfaces de usuario (IU) de los sistemas computacionales, son el espacio de contacto entre dos entidades: el usuario y el sistema de información. Una interfaz representa un conjunto de objetos, herramientas y representaciones visuales para gestionar la comunicación entre usuarios y computadoras.

Anteriormente las IU se mostraban como terminales alfanuméricas, por lo que la representación de la información era elemental. La aparición de las interfaces gráficas de usuario (GUI), hacen que en la actualidad el usuario tenga una mejor interación con los sistemas, además de que permiten el manejo de mayor número de elementos en su construcción [Pinninghoff, 2002].

En el área de la IA, las interfaces de usuario cubren múltiples modalidades incluyendo: lenguaje natural, gestos, gráficos y animación. Dependiendo de los requerimientos del usuario, estos elementos pueden actuar como agentes inteligentes que le permitan establecer lo que desea, determinando automáticamente las acciones requeridas al momento de su realización [Pinninghoff, 2002].

Un *agente virtual de interfaz* o un *agente de interfaz* se puede definir como un programa autónomo que juega un papel de intermediario entre un usuario humano y una computadora. Un agente de interfaz difiere de una interfaz ordinaria en que se espera que cambie su comportamiento y acciones de forma autónoma de acuerdo al comportamiento y situaciones del usuario humano en el proceso de interacción [Mase, 1997].

También se espera que el agente se mantenga de forma consistente, con una personalidad definida o un carácter aún cuando éste muestre varios comportamientos en diferentes situaciones dependientes de un contexto. Esto es importante cuando un agente se diseña como un agente de interfaz.

Se puede tener una aproximación con respecto a las Interfaces de Usuario Distribuidas, esto debido a la tecnología existente en la actualidad que permite construir interfaces que se puedan usar de forma colectiva entre diferentes usuarios, así como se hace con un chat, una misma interfaz es utilizada por diversos usuarios [Eriksson, 1999].

La tecnología e infraestructura de la Internet permite la implementación de sistemas expertos como servidores de conocimiento. De igual forma el interés de la IA en dar soporte a los servicios de navegación va en aumento. Diversos grupos de investigación trabajan en agentes inteligentes que ayuden a los usuarios a encontrar información y realizar tareas a través de la red. También se pueden encontrar agentes como sistemas expertos que incorporan conocimiento acerca de cómo encontrar servicios de red apropiados [Eriksson, 1999].

En este trabajo se plantea una interfaz multiagentes que permite entrevistar a uno o varios usuarios. Y por las características de la plataforma, se puede tener de forma distribuida.

Apéndice B

Clases en Jess

En este apéndice, se encuentra la aportación realizada para complementar la documentación existente de Jess. El Cuadro B.l es el descriptivo de las clases de Jess. El Diagrama de Clases de Jess se muestra en la Fig.B.l y las asociaciones de las clases de Jess se describen en el Cuadro B.2

B.l. Clases del paquete Jess

Clase	Descripción
Activation	Activación de una regla
BagFunctions	Funciones definidas por el usuario para la manipulación de variables "bags"
Binding	Liga- valores a las variables; internamente también liga variables a slots en hechos.
Console	Una consola gráfica para Jess
ConsoleApplet	Un applet que usa ConsolePanel
ConsolePanel	Un diálogo GUI básico de preguntas y respuestas
Context	Una ejecución context para Funcalls
Deffacts	Clase utilizada para representar deffacts
Deffunction	Clase utilizada para representar Deffunctions (funciones definidas
Defglobal	Clase utilizada para representar Defglobals
Defquery	Clase utilizada para representar Defqueries
Defrule	Clase utilizada para representar Defrules
Deftemplate	Clase utilizada para analizar, imprimir y representar deftemplates
DumpFunctions	Implementa las funciones bload y bsave de Jess

Fact	Un Fact es un ValueVector donde las entradas son datos slot en orden de declaración
Funcall	Una clase para analizar, armar e interpretar function calls
FuncallValue	Una clase para representar una function call de Jess almacenada en un Value
HasLHS	Clase padre de Defrules y Defqueries
Jesp	Analizador para Jess
JessEvent	Los JessEvents son usados por los fuentes JessEvent para transmitir información acerca de cosas interesantes que pasan en los
JessEventAdapter	Permite escribir manejadores JessEvent en Jess
LongValue	Clase que representa un long de Java
Main	Una interface de línea de comandos de Jess, también es desplegado en una ventana por la clase Console
MathFunctions	Funciones matemáticas para Jess
MiscFunctions	Algunas funciones misceláneas
MultiFunctions	Funciones para tratar con multifields
Node	(C) 1997 Ernest J.
Pattern	Representa un elemento simple condicional en una regla LHS
PredFunctions	Funciones predicado (es X del tipo Y?)
ReflectFunctions	Java Reflection para Jess
Rete	La máquina de inferencia
RU	Utilidades generales para Jess
StringFunctions	Implementa funciones de manipulación de String
Testl	Mantiene un solo test en un Patterna en el LHS de una regla
Token	Un Token es la unidad fundamental de comunicación en la red Rete
Value	Una clase que representa un tipo value en Jess
ValueVector	Una mini versión de Vector, sólo mantiene Values
Variable	Una clase para representar una variable Jess
ViewFunctions	Un visor gráfico de la red RETE para Jess

Cuadro B.l: Descripción de clases de Jess

B.l.l. Diagrama de clases de Jess

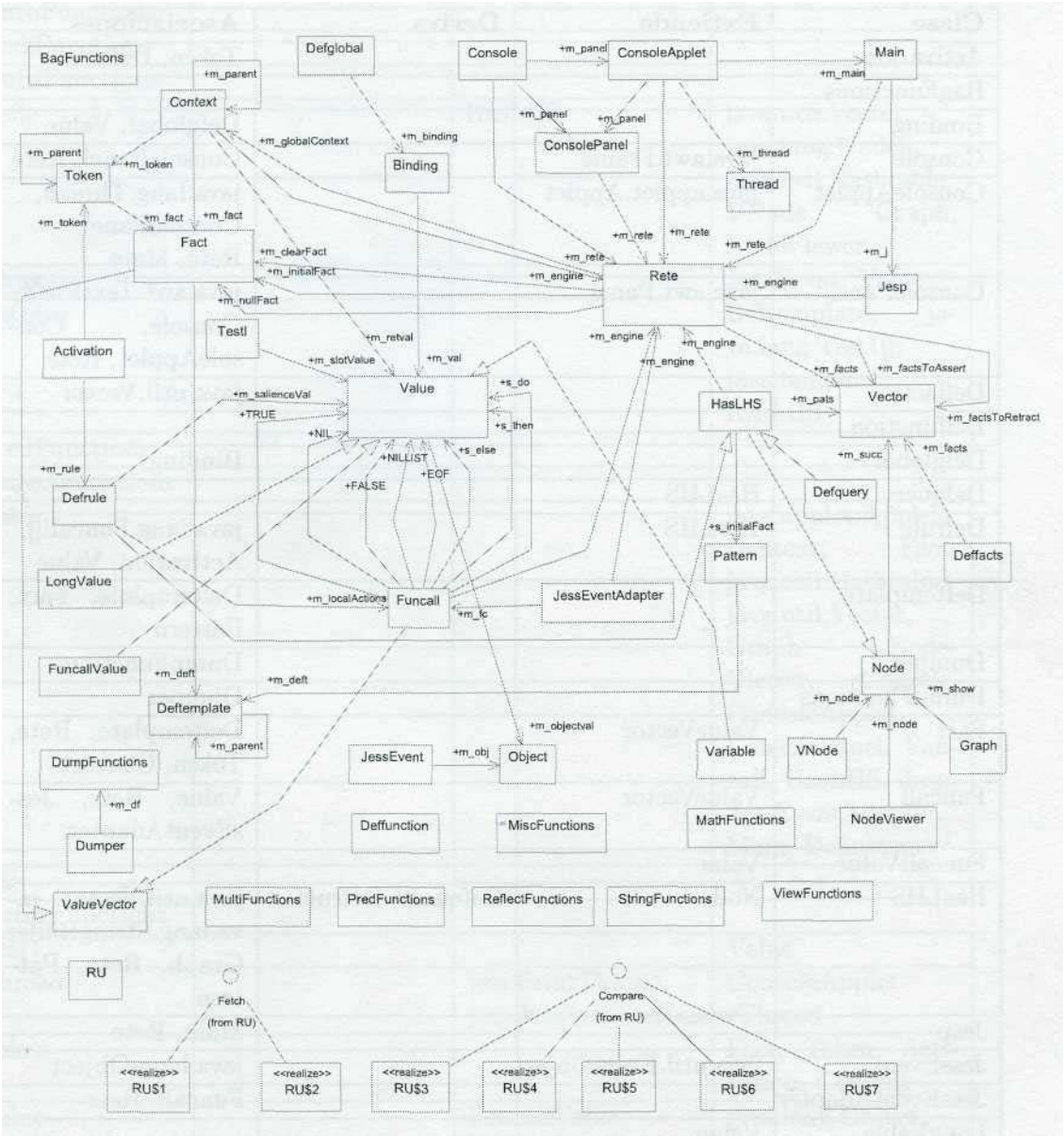

Figura B.l: *Diagrama de clases de Jess.*

B.1.2. Asociaciones del diagrama de clases de Jess

Clase	Extiende	Deriva	Asociaciones
Activation			Token, Defrule
BagFunctions			
Binding			Defglobal,Value
Console	java.awt.frame		ConsolePanel, Rete
ConsoleApplet	java.applet.Applet		java.lang.Thread, ConsolePanel, Rete, Main
ConsolePanel			java.awt.TextField, Console, ConsoleApplet, Rete
Deffacts			java.until.Vector
Deffunction			
Defglobal			Binding
Defquery	HasLHS		
Defrule	HasLHS		java.lang.Funcall[], Activation, Value
Deftemplate			Deftemplate, Fact, Pattern
Dumper			DumpFunctions
DumpFunctions			Dumper
Fact	ValueVector		Deftemplate, Rete, Token, Context
Funcall	ValueVector		Value, Rete, JessEventAdapter
FuncallValue	Value		
HasLHS	Node	Defquery, Defrule	java.util.Vector, java.lang.StringBuffer Graph, Rete, Pattern
Jesp			Main, Rete
JessEvent	Java.util.EventObject		java.lang.Object
JessEventAdapter			Funcall, Rete
LongValue			

Main			java.io.Reader, Jesp, ConsoléApplet, Rete
MathFunctions			
MishFunctions			
MultiFunctions			
Node		HasLHS	java.util.Vector, java.lang.Node[], java. util. Hashtable, VNode, Graph, NodeViewer
Object			JessEvent, Value
Pattern			Deftemplate, java.lang.Testl[], java.lang.int[], HasLHS
PredFunctions			
ReflectFunctions			
Rete			java.applet.Applet, Context, Fact, java.util.Hashtable, java.util. Vector, Graph, Node Viewer, Console, Console Applet, ConsolePanel, Funcall, HasLHS, Jesp, JessEventAdapter, Main, Definstance
RU			
StringFunctions			
Testl			Value
Thread		jess.PrintThread, jess.JessSystwem. ReaderThread	ConsoleApplet
Token			Fact, Token, Activation, Context
Value		FuncallValue, LongValue Variable	java.lang.Object, Binding, Context, Defrule, Funcall

ValueVector		Fact, Funcall	java.lang.Value[]
Variable	Value		
Vector			Deffacts, HasLHS, Node, Rete
ViewFunctions			

Cuadro B.2: Asociaciones en las clases de Jess

Bibliografía

[Bellifemine y Trueco, 2001] Bellifemine F., Trueco T., ***JADE Documentation on Line,*** CSELT and the Computer Enfineering Group of the University of Parma, Italy, http://sharon.cselt.it/projects/jade/, 2001.

[Blake y Merz, 1998] Blake, C. L., Merz, C. J. UCI Repository of machine learning databases. University of California, Department of Information and Computer Science, Irvine, CA. http://www.ics.uci.edu/ mlearn/MLRepository.html, 1998.

[Boissier et al, 2000] Boissier O., Beaune P., Sayettat C, Carron T., Gaultier F., Hannoun M., Proton H., Vercouter L.,**La Plate-forme MAST (Multi Agent System Toolkit),** Equipe SMA/Ecole Nationale Supérieure des Mines de Saing-Etienne, 2000.

[Bradshaw, 1999] Bradshaw J. M., ***An Introduction to Software Agents,*** in Software Agents, Bradshaw, J.M., (ed.) Cambridge, MA: MIT Press, 1997.

[Brenner, Zarnekiq y Wittig, 1998] Brenner W., Zarnekiq, R., and Wittig, H., ***Intelligent Agents: Foundations and Applications,*** Berlin: Springer-Verlag, 1998.

[Clearwater, Hogg y Huberman, 1992] Clearwater S.,Hogg T., Huberman B., *Cooperative* ***Problem Solving,*** Computation: The micro and Macro view, (ed) World Scientific, 1992.

[Colle, 1999] Colle, Raymond. ***Ciencias Cognitivas y Comunicación,*** Centro de Estudios Mediales, http://facom.udp.cl/CEM/bajada/index.php, 1999.

[Criado, 2000] Criado J., ***Introducción a los Sistemas Expertos,*** Universidad Complutense de Madrid, http://www.ingenieroseninformatica.org/recursos/-tutoriales/sist_exp/index.php, 2000.

[Cuena, 1998] Cuena, J., ***Los Sistemas multiagentes basados en el conocimiento: ¿ Una posible alternativa para la Moderna Ingeniería del Software ?,*** Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 1998, 6(8):pp. 85-95.

[d'Inverno, Luck y Wookdridge, 1997] d'Inverno, M., M. Luck and M. Wooldridge. ***Cooperation Structures.*** In Proceedings of 15. Int. Joint Conference on AI, Nagoya, Japan, 1997, pp. 600-605.

[Decker et al., 1995] Decker K., Lesser V., Nagendra M. V., Wagner T., ***MACRON: An Architecture for Multi-agent Cooperative Information Gathering,*** In CIKM Conference, Workshop on Intelligent Information Agents, 1995.

[Doran, Franklin y Jennings, 1997] Doran J.E., Franklin S., Jennings N.R., and Norman T.J. ***On cooperation in multiagent systems***. Knowledge Engineering Review, 1997, 12(3):309-314.

[Durfee, 1995] Durfee, E. In M., ***Distributed Artificial Intelligence,*** In Arbib (ed.) The Handbook of Brain Theroty and Neural Networks, MIT Press, 1995.

[Elizalde, 2000] Elizalde V. G., ***Conceptos avanzados sobre interfaces de usuario,*** http://www.fismat.umich.mx/ elizalde/tesis/node46.html, 2000.

[Eriksson, 1999] Eriksson H., ***Expert Systems as Knowledge Servers,*** Intelligent Systems IEEE, http://www.computer.org/intelligent/exl996/x3014abs.htm, 1996, 11(3).

[Etzioni, Lesh y Segal, 1994] Etzioni, O., Lesh, N., and Segal, R. ***Building softbots for UNIX.*** In Etzioni, O., (ed.) *Software Agents - Papers from the 1994 Spring Symposium (Technical Report)* AAAI Press, 1994.

[Ferber y Gutknecht, 1998] , Ferber, J., Gutknecht, O., ***A meta-model for the analysis and design of organizations in multi-agent systems,*** Laboratorio de Informática, Robótica y Micro-electrónica, Universidad de Montpellier, Francia, 1998.

[Finin, Labrou y Mayfield, 1995] Finin T., Labrou Y. y Mayfield J., ***KQML as an agent communication language,*** Computer Science and Electrical Enfineering, University of Maryland Baltimore County, Baltimore MD USA, 1995.

[Finin y Weber, 1994] Finin T., Weber J., ***Specification of the KQML Agent-Communication Language,*** The DARPA Knowledge Sharing Iniciative External Interfaces Working Group, 1994.

[FIPA, 2000] Foundation For Intelligent Physical Agents, ***FIPA-ACL Message Structure Specification***. http://www.fipa.org/, 2000.

[Fischer y Chaib-draa, 1999] Fischer K., Chaib-draa B., ***A Simulation Approach Based on Negotiation and Cooperation Between Agents: A Case Study***. IEEE Transaction on Systems, Man and Cybernetics, Part C: Applications and Reviews, 29(4), 1999.

[Friedman, 2000] , Friedman E. J., ***Jess, The Java Expert System Shell,*** Distributed Computing Systems, Sandia National Laboratories, 2000.

[García y Ossowski, 1998] , García-Serrano A., Ossowski, S. ***Inteligencia Artificial Distribuida y Sistemas Multiagente,*** Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 1998, 6(2): pp. 12-23.

[Gardner, 1987] , Gardner, H. ***La nueva ciencia de la mente,*** (ed) Paidós, Barcelona, 1987.

[Genesereth y Ketchpel, 1994] Genesereth, M. R. and Ketchpel, S. P. ***Software Agents. Communications of the ACM,*** 1994.

[Giarratano y Riley, 2001] Giarratano J., Riley G. *Sistemas Expertos principios y programación,* Tercera ed., International Thomson Editores, 2001.

[Gilbert, 1997] Gilbert D., *Intelligent* ***Agents: The Right Information at the Right Time***. IBM Corporation, 1997.

[Gruber, 1993] Gruber T. R., *A* ***Translation Approach to Portable Ontology Specifications,*** Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1993.

[Gutknecht y Ferber, 1997] Gutknecht O., Ferber J., ***MadKit: Organizing heterogeneity with groups in a platform for múltiple multi-agent systems,*** Laboratoire D'Informatique, de Robotique et Micro-Electronique de Montpellier. Unité Mixte CNRS - Université Montpellier II C 09928, France, 1997.

[Gutknecht y Ferber, 2000] Gutknecht O., Ferber J., ***The MADKIT Agent Platform Architecture,*** Laboratoire D'Informatique, de Robotique et Micro-Electronique de Montpellier. CNRS - Université Montpellier II, France, 2000.

[Haugeneder y Steiner, 1998] Haugeneder H., Steiner D., ***Co-operating Agents: Con-cepts and Applications,*** (ed.) *"Agent Technology: Foundations, Applications, and Markets",* Jennings N. R., Wooldridge M. J., 1998.

[Iglesias, Garijo y González, 1998] Iglesias C. A., Garijo J., Gonzáles J.C., ***Metodologias orientadas a agentes,*** Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, Monográfico Inteligencia Artificial Distribuída y Sistemas Multiagentes, 1998, 6(2).

[Iglesias, 1999] Iglesias C. A. ***Sistemas Basados en Conocimiento,*** Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, http://www.gsi.dit.upm.es/ cif/cursos/ssii/, 1999.

[IKV++, 2001] ***Grasshopper Basics and Concepts Paper, Release 2.2,*** Grasshopper 2 The Agent Platform, IKV++ Technologies Ag, Germany, http://www.grasshopper.de, 2001

[Jackson, 1999] Jackson P., ***Introduction to Expert Systems***. Third Ed., Addison Wesley Longman Limited, 1999.

[Jennings y Wooldridge, 1998] Jennings N. R., Wooldridge M. J. ***Agent Technology, Foundations, Applications, and Markets,*** 1998.

[Jeon, Petrie y Cutkosky, 2000] Jeon H., Petrie C, Cutkosky M. R., ***JATLite: A Java Agent Infrastructure with Message Routing,*** Stanford Center for Design Research (CDR), 2000.

[Keung-Chi, 1990] Keung-Chi Ng, Bruce Abramson. ***Consensus in a Multi-Expert System***. Department of Computer Science, University of Southern Califórnia, Los Angeles, CA. 1990.

[Lespérance et ai., 1996] Lespérance, Y., Levesque, H. J., Lin, F. y Marcu, D. Foundations of a Logical Approach to Agent Programming. Springer-Verlag, 1996.

[Maes, 1997] Maes P. ***CHI97 Software Agents Tutorial,*** Conference on Human Factors in Computing Systems, MIT Media Laboratory, 1997.

[Marcenac, Leman y Giroux, 1996] Marcenac P., Leman S., Giroux S. ***Cooperation and Conflicts Resolution in MultiAgent Systems,*** in Proceedings of the ACM South-East Conference, Tuskegee, AL, 1996, pp.289-291.

[Mase, 1997] Mase K**.** ***Aspects of Interface Agents: Avatar, Assistant and Actor,*** IJCAI'97 Workshop on Animated Interface Agents, 1997.

[Minsky, 1968] Minsky M. ***Semantic Information Processing,*** (ed.) Cambridge, MA: MIT Press, 1968.

[Moulin y Chaib-draa, 1996] Moulin B., Chaib-draa B. ***An Overview of Distributed Artificial Intelligence. In Foundations of Distributed Artificial Intelligence,*** (ed.) G. M. P. O'Hare and N. R. Jennings, New York: Wiley, 1996.

[Neville y Jensen, 2003] Neville J., Jensen D., ***Collective Classification with Relational Dependency Networks***. 2nd Workshop on Multi-Relation Data Mining MRDM 2003. http://www-ai.ijs.si/SasoDzeroski/MRDM2003/, 2003.

[Nwana, 1996] Nwana H. *&¿* Ndumu Divine. ***An Introduction to Agent Technology,*** Intelligent Systems Research Advanced Applications and Technology. Report, 1996.

[Ortega et al., 1997] Ortega M., Bravo J., Prieto M., de Lara J**.** ***Groupware y Educación,*** Departamento de Informática, Universidad de Castilla - La Mancha, publicación en la revista *"Enseñanza y Tecnología",* (ed.) Asociación para el Desarrollo de la Informática Educativa (ADIÉ), http://chico.inf-cr.uclm.es:8080/adie/index.html, 1997.

[Ossowski y Serrano, 1998] Ossowski S., García-Serrano A. ***La coordinación en Sociedades Artificiales de Agentes,*** Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 1998, 6(8): pp. 46-56.

[Papazoglou y Schlageter, 1998] Papazoglou M. P., Schlageter G., ***Cooperative Information Systems, Trends and Directions,*** Academic Press, 1998.

[Patil et al., 1992] Patil, R. S., R. Fikes, P.F. Patel-Schneider, D. McKay, T. Finin, G. Thomas y R. Neches. *The* ***DARPA knowledge sharing effort progress report,*** Proceedings of the Third Int. Conf. on Principies of Knowledge Representation and Reasoning, 1992.

[Pavón y Gómez, 2003] Pavón M. J., Gómez S. J., ***Agent Oriented Software Engineering with INGENIAS,*** In: Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS), Lecture Notes in Computer Science 2691, (ed.) Springer Verlag, 2003, pp. 394-403.

[Pinninghoff, 2002] Pinninghoff M. A., ***Por qué vale la pena interesarse en IA?,*** Inteligencia Artificial, Universidad de Concepción, Departamento de Ingeniería Informáti - ca y Ciencias de la Computación, Chile, http://www.info.udec.cl/ intartif/, 2002.

[Rendón, 2000] Rendón G. A. ***El Lenguaje Unificado de Modelado (UML),*** Universi - dad del Cauca, Facultad de Ingeniería Electrónica y Telecomunicaciones Departamento de Conmutación. Popayán, http://www.inf.udec.cl/ intartif/porque.htm, 2000.

[Riquenes y Alba, 1997] Riquenes A., Alba E., ***Clasificación Colectiva: Una alternativa válida para la clasificación de objetos.*** II Taller Iberoamericano de Reconocimiento de Patrones, La Habana, Cuba. Memorias TIARP97, la. ed. 1997, pp. 179-188, ISBN: 968-29-9892-1.

[Russel y Norvig, 1995] Russel S., Norvig P. ***Artificial Intelligence — A Modern Approach.*** New Jersey, Prentice-Hall, 1995.

[Saadoun, 1997] Saadoun M., ***El proyecto groupware. De las técnicas de dirección a la elección de la aplicación groupware,*** Ediciones Gestión 2000 S.A., Barcelona, 1997.

[Salcedo, 2000] Salcedo L. P. Artículo: ***Ingeniería de software educativo, teorías y metodologías que la sustentan,*** Revista de Universidad de Concepción, Departamento de Ingeniería Informática y Ciencias de la Computación. http://www.inf.udec.cl/revista/psalcedo.htm, Chile, 2000.

[Samper, 2002] Samper M. J. ***Sistemas Expertos. El conocimiento al poder***. Universidad de Granada, http://www.psycologia.com/articulos/ar-jsamper01 _2.htm, 2002.

[Shoham, 1993] Shoham, Y. ***Agent Oriented Programming,*** Artificial Intelligence, 6: pp. 51-92, 1993.

[Sierra et al., 1998] Sierra C, Jennings N. R., Noriega P., Parsons S., A framework for argumentation-based negotiation. In M.P. Singh, A Rao, and M. J. Wooldridge, ed. Proc. ATAL-97, pp. 177-192, Berlin, Germany, 1998.

[Soto y Núñez, 2003] Soto C. R., Núñez E. G., ***Soft Modelling of Financial Time Se -- ries.*** Proceedings of the IASTED International Conference MODELLING AND SIMULATION, Palm Springs, California, USA, 2003, pp. 537-542, ISBN: 0-88986-337-7.

[Stone y Veloso, 1997] Stone P., Veloso M. ***Multiagent Systems: A Survey from a Machine Learning Perspective,*** Computer Science Department, Carnegie Mellon University, 1997.

[Weiss, 1999] Weiss G. ***Multiagent Systems - A modern Approach to Distributed Artificial Intelligence*** (ed.) Gerhard Weiss. The MIT Press Cambridge, Massachussetts London, England, 1999.

[Winstanley, 1987] Winstanley G. ***Program Design For Knowledge Based Systems,*** Sigma Press Wilmslow, UK, 1987.

[Wooldridge, 1995] Wooldridge M. J. ***Intelligent Agents: Theory and Practice,*** Department of Computing, Manchester Metropolitan University. Jennings, Nicholas. Department of Electronic Engineering, Queen Mary.& Westfield College, 1995.

[Wooldridge y Jennings, 1994] Wooldridge, M. J., Jennings N. R. ***Agent Theories, Architectures, and Languages: A Survey,*** Departament of Computing Manchester Metropolitan University Chester Street, 1994.