UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

POSGRADO EN CIENCIA E INGENIERIA
DE LA COMPUTACION

DESARROLLO DE UNA ARQUITECTURA
PARA SISTEMAS EXPERTOS EN WEB

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

P R E S E N T A:

JOSE GUADALUPE GOMEZ FUENTES

DIRECTOR DE TESIS:
DR. NICOLAS KEMPER VALVERDE

MEXICO, D. F. 2001

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

UNAM

POSGRADO EN CIENCIA E INGENIERIA DE LA COMPUTACION.

DESARROLLO DE UNA ARQUITECTURA PARA SISTEMAS EXPERTOS EN WEB

TESIS

QUE PARA OBTENER EL GRADO DE:
MAESTRO EN CIENCIAS

PRESENTA:

JOSE GUADALUPE GOMEZ FUENTES

DIRECTOR DE TESIS: DR. NICOLAS KEMPER VALVERDE

MEXICO D.F 2001

Capitulo IV
Implementación del sistema

Introducción.

La Web se ha convertido en una herramienta universal de comunicaciones, montar software y programas de aplicación en la Web, permite que esté disponible a usuarios de todo el mundo, y si el conocimiento sobre cualquier materia puede ser directamente proporcionado a los usuarios, entonces porqué no poner los sistemas expertos en la Web. Desafortunadamente, la Web no fue creada para aplicaciones de este tipo, hasta hace poco, la manera en la cual funcionaban la Web y los navegadores presentaba dificultades para realizar las acciones requeridas por los sistemas expertos. Sin embargo, nuevas tecnologías vienen al frente para cambiar esto situación.

Los sistemas expertos han demostrado su gran utilidad en los últimos diez años, se han aplicado a muchas áreas tanto de la investigación como en la industria, la tendencia hacia aplicaciones de Internet no se ha quedado atrás, cada día se están buscando mejores técnicas de programación en la Web, para así, extender las posibilidades de nuevas aplicaciones como: agentes y programas inteligentes que tienen vida en un ambiente real como lo es Internet.

Los primeros sistemas expertos habilitados para correr en Web aparecieron en 1996. Hay dos maneras para habilitar en la Web la máquina de inferencia de un sistema experto: (1) correr el sistema en un servidor o (2) en un cliente (computadora local del usuario). A medida que la Web y las tecnologías asociadas con los lenguajes de programación se desarrollen, más opciones estarán disponibles para cada una.

Uno de los primeros sistemas expertos habilitados para correr en Web apareció en 1996. En este sistema experto, para preguntar al usuario y obtener resultados, la interfaz de Web fue agregada a la Máquina de Inferencia (MI) de un sistema experto establecido de manera local. Este requería correr la MI, mediante un programa CGI en el servidor, para procesar los datos en páginas html y para realizar otras tareas. El principal problema con CGI, y la mayoría de las arquitecturas basadas en servidor, para sistemas expertos es "cómo guardar el estado del sistema". El proceso de guardar la interacción del usuario y cualquier información del procesamiento acerca de dicho usuario.

Posteriormente han habido varios avances para la ejecución de programas del lado del cliente. Java ha sido el más popular y exitoso. Java provee un marco de trabajo para crear programas que corran en la máquina del cliente, simplemente incluyendo el código apropiado del programa en el código html de una página Web. El programa en Java, tiene acceso a la Web, puede cargar cualquier archivo de datos e imágenes requeridas por el sistema experto.

El objetivo de este trabajo es desarrollar una arquitectura para poder montar los sistemas expertos en la Web. Para ello primero, se determinarán los recursos necesarios tanto en hardware como en software para desarrollar una arquitectura

que permita el desarrollo de sistemas expertos por un lado, y por otro, montarlos en la Web para que estén a disposición de usuarios en todo el mundo.

Una vez establecidos los recursos necesarios, se procederá a desarrollar la arquitectura planteada y a establecer el alcance para que un usuario pueda acceder a través de un navegador a los programas desarrollados con dicha arquitectura.

En el primer capítulo se da un panorama general de los sistemas expertos, sus principales características y estructura general; posteriormente, se describe a la Internet y la Web, sus orígenes, cómo funciona y por qué se ha convertido en uno de los inventos más importantes del siglo XX. Por último, los principales medios para montar programas y aplicaciones en la Web.

En el segundo capítulo se hace el planteamiento del problema, y con base en ello, se establecen cuáles serán las características y los requisitos de la arquitectura para lograr cumplir los objetivos.

En el tercer capítulo se describe el marco teórico de la Inteligencia Artificial (IA) y los Sistemas Expertos (SE), se explica la arquitectura de éstos y el funcionamiento de cada uno de los elementos que los constituyen; al final se hace un énfasis en los sistemas expertos basados en reglas, que es la técnica de desarrollo, en que se basan los sistemas que serán desarrollados por nuestra arquitectura.

En el cuarto capítulo se describe la implementación de la arquitectura, primero se explica cómo se estructuró el hardware para estar conectado a la Web, y en segundo lugar, se presenta el software desarrollado para complementar el funcionamiento de la arquitectura propuesta y cumplir con el objetivo de mi trabajo.

Por último, se dan las conclusiones y resultados obtenidos de este trabajo.

Capítulo I

Panorama general de Sistemas Expertos y Web.

Introducción.

De todas las áreas de la Inteligencia Artificial, los sistemas expertos es una de las más aplicadas, y sin embargo aún constituyen un gran reto en las investigaciones de la computación especialmente al tratar de integrarlo a nuevas tecnologías como el Internet.

Un sistema experto, es un sistema que emplea conocimiento humano capturado en una computadora para resolver problemas que normalmente requieren de la expertise[1] humana; si estos sistemas están bien desarrollados imitan el proceso de razonamiento que los expertos usan para resolver problemas específicos.

Ultimamente estos sistemas en ocasiones han funcionado mejor que cualquier experto humano al realizar análisis en un área especifica que generalmente se conoce como **Dominio**. Esta posibilidad ha tenido un impacto significante tanto en profesionales (como analistas financieros, abogados, analistas fiscales etc.), como en grandes organizaciones. El propósito de este capítulo es introducir los fundamentos de los sistemas expertos, una breve historia, seguida de sus capacidades y estructura y finalmente sus beneficios y limitaciones; además del uso de los servicios de World Wide Web como una opción potencial para la aplicación de estos sistemas de forma masiva o multiusuario.

1.1 Historia.

Los sistemas expertos fueron desarrollados por la comunidad de Inteligencia Artificial (IA), a mediados de los años 60, en esta época los investigadores creían que algunas leyes del razonamiento combinadas con el potencial de las computadoras podrían producir un programa con desempeño super humano o experto [8]. Un primer intento en el objetivo de esta investigación fue el desarrollo del Solucionador de Problemas de Propósito General (SPPG). Este fue un procedimiento desarrollado por Newell y Simon de una máquina teórico lógica que fue un intento por crear una computadora inteligente y puede ser considerado como el predecesor de un sistema experto. El SPPG trabaja con una serie de pasos necesarios para cambiar una cierta situación inicial, a un estado o meta deseada (o situación final) [8].

Para cada problema a resolver, al SPPG se le da: (1) un conjunto de operadores, que pueden cambiar una situación de varias maneras, (2) una descripción de que precondiciones necesita cada operador antes de ser aplicado y (3) una lista de

[1] Expertise: es un término en inglés, que se usa para definir o hacer referencia, a la experiencia y conocimiento de una persona especializada en un dominio.

postcondiciones que serán verdaderas después de que el operador ha sido usado; en términos de un sistema experto, esto es una base de conocimiento (BC).

El cambio de propósito general a propósito específico ocurrió a mediados de los años 60 con el desarrollo de DENDRAL, por E. Feigenbaun en la Universidad de Stanford y seguido por el desarrollo de MYCIN. Fueron dos de los primeros sistemas expertos con aplicaciones reales, DENDRAL servía para deducir información sobre estructuras químicas y MYCIN era un sistema de diagnóstico para enfermedades de la sangre.

Así, en los años subsecuentes se desarrollaron más sistemas expertos, y reconociendo el papel central del conocimiento en estos sistemas, los investigadores en Inteligencia Artificial, trabajaron en el desarrollo de teorías de representación del conocimiento y procedimientos generales para la toma de decisiones.

1.2 Definición de los Sistemas Expertos.

Los sistemas expertos son programas de computadora que tratan de imitar los procesos de razonamiento y conocimiento de los expertos en la solución de un tipo específico de problemas. Son usados más que cualquier otra tecnología de Inteligencia Artificial, además de que son de gran interés en las organizaciones por su potencial para obtener la productividad y aumentar las fuerzas de trabajo en muchas áreas de especialidad donde los expertos son difíciles de encontrar o retener [2].

Para comprender mejor esto, primero veamos como trabajan los expertos humanos. Los expertos humanos tienden a especializarse en un área relativamente específica para la solución de problemas o realización de ciertas tareas, esto se conoce como **Dominio del experto**. Típicamente, los expertos humanos poseen las siguientes características: *ellos resuelven problemas rápidamente y con gran exactitud, explican qué, y a veces cómo lo hacen, inclusive juzgan la confiabilidad de sus propias conclusiones, saben cuándo están cortos en la solución, y se comunican con otros expertos.* Ellos también pueden aprender de la experiencia, combinando sus puntos de vista para satisfacer un problema y transferir conocimiento de un dominio a otro. Finalmente usan herramientas como reglas de dedo, modelos matemáticos y simulaciones detalladas para soportar mejor sus decisiones.

Un sistema experto técnicamente es un programa de computadora que soluciona problemas complicados que de otra manera exigirían ampliamente la pericia humana. Para lograr esto, se simula un proceso de razonamiento humano mediante la aplicación específica de conocimientos e inferencias [2].

Internamente un sistema experto ideal se puede caracterizar como un sistema que comprende:

- Amplio conocimiento específico a partir del campo de interés.
- Aplicación de técnicas de búsqueda.
- Soporte para análisis heurístico.
- Habilidad para inferir nuevos conocimientos a partir de los ya existentes.
- Procesamiento de símbolos.
- Capacidad para explicar su propio razonamiento.

De todas, el conocimiento es el mejor recurso y es importante capturar el conocimiento de tal manera que otras personas puedan usarlo fácilmente. Los expertos humanos pueden enfermarse o no estar disponibles y entonces el conocimiento no siempre está disponible cuando se necesita; los libros y manuales pueden tener un poco de conocimiento, pero muchas veces, dejan el problema de una aplicación en particular al criterio del lector. Los sistemas expertos pueden proveer un medio directo de la aplicación de la expertise, ya que permite que el conocimiento de uno o más expertos sea capturado y almacenado en una computadora, así el conocimiento puede ser usado por quien lo requiera. El propósito de un sistema experto no es reemplazar a los expertos humanos, sino simplemente hacer más amplios y disponibles sus conocimientos y experiencia. Un sistema experto permite a otros incrementar su productividad, mejorando la calidad de sus decisiones y solución de problemas cuando un experto no está disponible.

Existen varios componentes del conocimiento, que dan origen a la habilidad experta de su desempeño [2]. Estos se pueden ver generalmente como:

- *Hechos.* Son declaraciones que relacionan algunos elementos de la realidad con referencia al área específica. Por ejemplo:
  La leche es blanca.
  Un Boeing 747 vuela sin problemas con tres motores.

- *Reglas de procedimiento.* Son reglas bien definidas e invariables que describen secuencias fundamentales de eventos y relaciones relativas al área. Por ejemplo:
  Siempre verifique el tránsito vehicular antes de entrar a una autopista.
  Si el altímetro señala el nivel de vuelo, el medidor de velocidad vertical debe marcar cero.

- *Reglas heurísticas.* Son reglas generales en forma de opiniones o reglas empíricas que sugieren procedimientos que se pueden seguir cuando no existen disponibles reglas de procedimiento invariables. Estas reglas son aproximadas y han sido generalmente acuñadas por un experto a través de años de experiencia. Por ejemplo:

  Si una sierra parece estar bien, pero aún así no arranca, afloje la tensión de la cadena.

Es mejor intentar un aterrizaje de emergencia bajo condiciones controladas que volar en condiciones desconocidas.

El uso de heurísticas contribuye grandemente a la potencia y flexibilidad de los sistemas expertos y tiende a distinguirlos aún más del software tradicional.

1.3 Estructura de los Sistemas Expertos.

Los sistemas expertos se componen de dos partes principales: un Ambiente de desarrollo y uno de consulta; el primero es usado por el constructor de sistemas expertos para hacer los componentes e introducir el conocimiento a la base de conocimiento; el segundo es el ambiente usado por el no experto (usuario) para obtener conocimiento o recomendaciones [2]. Los componentes de un sistema experto son:

- Subsistema de adquisición de conocimiento (solo en Ambiente de Desarrollo)
- Base de Conocimiento (BC).
- Máquina de Inferencia (MI).
- Lugar de trabajo.
- Interfaz de Usuario (IU).
- Subsistema de explicación
- Subsistema de refinamiento. (solo en Ambiente de Desarrollo)

1.4 Estado actual de los Sistemas Expertos, capacidades y limitaciones.

Actualmente los sistemas expertos se emplean para ejecutar una variedad muy complicada de tareas que en el pasado solamente podían llevarse a cabo por un número limitado de personas expertas e intensamente entrenadas a través de la aplicación de técnicas de Inteligencia Artificial (IA) [7], los sistemas expertos captan el conocimiento básico que permite a una persona desempeñarse como un experto frente a problemas complicados. Los sistemas expertos pueden ser usados por:

- Usuarios no expertos para improvisar sus capacidades en la solución de problemas.
- Usuarios expertos como un asistente de conocimiento.

Los sistemas expertos actuales se han clasificado en tres clases:

Asistente. Es un pequeño sistema basado en conocimientos que realiza un subconjunto de una tarea experta, es valioso económicamente pero limitado técnicamente. Muchos "asistentes" en computadoras personales pertenecen a este tipo.

Colega. Es un sistema de tamaño mediano basado en conocimientos que realiza una parte significativa de una tarea experta, éstos sistemas se implementan tanto

en computadoras personales, como en instalaciones más grandes por ejemplo: estaciones de trabajo especializadas y grandes computadoras convencionales.

Experto. Es un sistema grande basado en conocimiento que se acerca al nivel del desempeño del experto, dentro de un dominio dado. Los expertos, normalmente se implementan en potentes instalaciones, utilizando herramientas complejas de desarrollo.

En los últimos años, el término sistema experto se ha aplicado en forma general a todas las anteriores clases de sistemas; se les llama sistemas expertos porque dependen de un experto como fuente de conocimiento y se desarrollan empleando técnicas basadas en conocimientos.

Es importante reconocer que aunque, los sistemas expertos de la actualidad sean potentes y muy útiles, existe un límite muy determinado para las capacidades en el estado del arte actual. Los sistemas típicos tienen las siguientes limitaciones:

- Los conocimientos se obtienen de un número pequeño de especialistas.
- La aplicación está orientada a un área específica limitada o a un pequeño conjunto de áreas.
- El área de aplicación debe tener poca necesidad de razonamientos temporales o espaciales.
- La tarea no depende del empleo de un gran volumen de conocimientos generales o de sentido común.
- El conocimiento que se requiere para ejecutar la tarea es razonablemente completo, correcto y estable.

Aunque tales limitaciones existen, hay muchas áreas que son apropiadas para la aplicación de los sistemas expertos.

1.5 Internet.

Hoy en día todos estamos familiarizados con la Internet, la mayoría de la gente la usa y en los últimos años ésta y una de sus principales aplicaciones la Web, se han convertido en un elemento de primera necesidad para muchas personas del mundo.

La red Internet es el resultado de comunicar varias redes de computadoras, y dentro de ella se pueden enviar o recibir archivos desde o hacia cualquier parte del planeta, encontrar información de cualquier tema: desde cómo hacer un pastel, hasta los últimos avances en física nuclear, también se pueden realizar compras electrónicas mediante una tarjeta de crédito, tener videoconferencias con colegas estudiantes en una Universidad de Argentina, o asistir a un concierto de U2 en Irlanda mientras te encuentras en la ciudad de México [10].

La forma de operación de la Internet se muestra en la figura 1.1, como se puede observar cientos de servidores, ó tal vez miles están conectados a la Internet. Los servidores son generalmente parte de una red de área local o también pueden ser varios servidores pertenecientes a una red amplia. Las aplicaciones son muy variadas, investigación, comercial, educación.

Nuestro objetivo es tener presencia en la Internet, como puede observar en la figura 1.1, un servidor Web para sistemas expertos formará parte de esta inmensa red de computadoras.

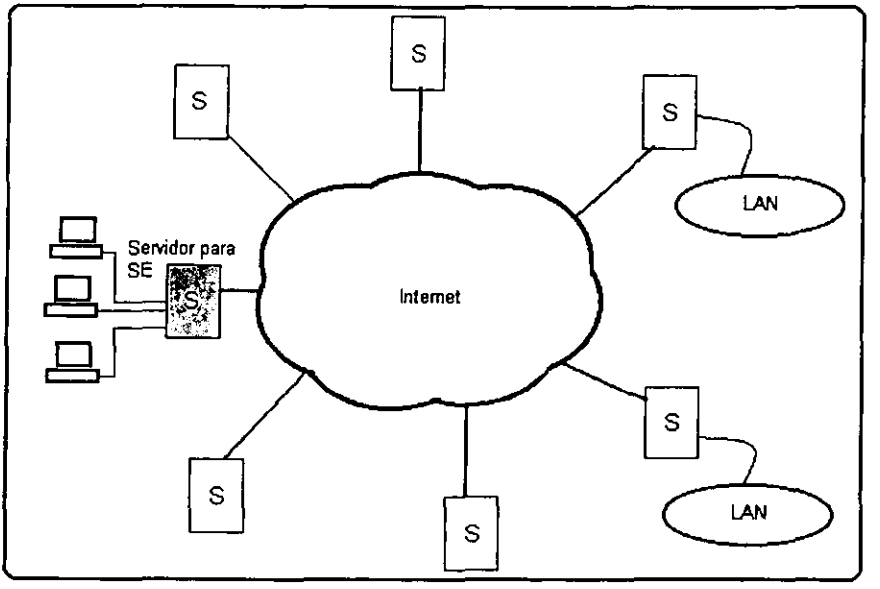


Figura 1.1 Internet

Lo mágico y excepcional de Internet, es que no tiene dueño, a nadie hay que pedirle permiso para publicar algo en ella y la mayoría de las páginas o servicios son gratuitos, además desde tu modesta PC 486 te puedes conectar a un poderoso servidor de UNIX, a un típico servidor Windows NT, a una Machintosh u otra PC.

Por otro lado, se calcula que a diciembre de 1998, la cantidad de sitios Web era de 5,000,000, pero diario, a cada instante, se unen más. La actual Internet es el antecedente directo de lo que se ha denominado "Super-Carretera de la Información" que está en desarrollo y muy pronto cambiará definitiva y permanentemente nuestra forma de vida. Dejará de ser solamente una fuente de información en tu computadora y se convertirá en un elemento indispensable para estar en contacto con el mundo, debido a que, nuestra terminal de Internet estará

conectada a todos los aparatos electrónicos de nuestra casa, por ejemplo a nuestra televisión.

Como se mencionó antes, Internet es un conjunto de redes de computadoras interconectadas que comparten un protocolo de comunicaciones (TCP/IP) [10]. Su límite, es el que tienen las redes que lo integran.

El medio de comunicación de estas redes de computadoras son las líneas telefónicas. Se estima que está compuesta por más de 100.000 redes y que la utilizan más de 100 millones de usuarios. En Internet se encuentran redes corporativas de Universidades, gobiernos, empresas privadas y de particulares. Nació en 1960 en el departamento de defensa de los Estados Unidos de América como un proyecto de investigación en los ámbitos militar y científico; En la década de los 80, se convirtió en una red puramente científica, especialmente dentro de la comunidad universitaria. A finales de los años 90, podemos decir que es una red en materia comercial, científica y de educación. En esta última década, gracias a los avances tecnológicos y a la incorporación de empresas dentro de la red, Internet se ha convertido en un auténtico fenómeno social que alcanza ya a millones de usuarios en todo el mundo [10].

En Internet no hay ningún amo, ni tan solo una única autoridad que gobierne la red. Esto se debe a que, hasta hace muy poco tiempo, la red estaba reservada únicamente a la actividad científica y por tanto, las organizaciones que ayudaban a la gestión de la red no eran lucrativas. Hoy en día la red ya es muy comercial y estas mismas organizaciones siguen "gobernando". Básicamente hay dos organizaciones [10]:

- La ISOC (*Internet Society*), organización profesional internacional que fomenta la evolución, estandarización y divulgación de técnicas y tecnologías.
- La IETE (*Internet Engineering Task Force*), grupo voluntario de ingeniería dentro de Internet. Produce diferentes grupos de trabajo y documentos. Esta entidad es la que marca las pautas de evolución de la red.

Internet no tiene límites, hay que explotarlo y aprender de él.

1.5.1 Beneficios del protocolo TCP/IP.

La característica fundamental del protocolo TCP/IP es que se envían mensajes empaquetados (*packets*) en una especie de envoltura digital, este mensaje es puesto en un paquete IP (*Internet Protocol*) y es enviado por la computadora fuente, y ella misma se encarga de hacer que el mensaje llegue en óptimas condiciones. Sí existe un problema en una parte de la red, la computadora fuente se percata y busca otro camino, pues lo único que le interesa es que el mensaje llegue íntegro, no importa cómo o por dónde, solo que llegue; es decir la probabilidad de que se pierda una señal bajo TCP/IP es muy baja, pues la computadora que envía la información no descansa hasta encontrar un camino adecuado por el cual viaje dicha información. Por otro lado, mediante el protocolo

TCP/IP quedaron atrás todos los problemas de incompatibilidad entre diferentes plataformas de computadoras, mediante TCP/IP nos podemos conectar a diferentes computadoras con diferentes sistemas operativos [10].

1.6 La World Wide Web.

Lo que veremos a continuación es el servicio del puerto 80 de TCP, la mundialmente famosa World Wide Web, conocido normalmente por el nombre de Web.

1.6.1 Definición.

El World Wide Web (Web) es uno de los múltiples servicios que ofrece Internet, debido a su facilidad de uso, su interfaz gráfica y su relativa interactividad se ha convertido en el servicio preferido de los usuarios de Internet [10].

El servicio Web se basa en la transmisión de hipertexto a través de la red y el cual es interpretado por los browsers o navegadores de Internet. Los navegadores más populares son Netscape Navigator y Microsoft Internet Explorer.

El hipertexto es una clase especial de texto que contiene instrucciones específicas para incrustar dentro del texto imágenes, ligas o hipervínculos a otros recursos Web dentro de la red, así como para dar formato al texto. El hipertexto utilizado en el Web se llama HTML (*Hiper Text Markup Language*) Un ejemplo clásico de HTML sería el siguiente:

```
<html>
<head>
<title>Ejemplo de Hipetexto</title>
<body>
</head>
¡Hola mundo!
<img src=http://www.servidor.com/nombredelaimagen.gif>
</body>
</html>
```

Es importante señalar, que el usuario de Internet no ve el código HTML, sino lo que ve es el documento ya formateado por el HTML.

A cada una de las instrucciones dentro del HTML se les llama tags o etiquetas y sirven para dar formato al documento y para hacer referencia a otras páginas, imágenes, o inclusive sonidos y videos. Podemos identificar fácilmente las etiquetas porque se encuentran encerrados entre los signos <> así, en el ejemplo anterior encontramos las etiquetas *<html>*, *<head>*, *<title>* *<body>* e *<img>*.

En el mismo ejemplo, la etiqueta *<html>* indica al navegador, que está iniciando un documento HTML; la etiqueta *<head>* sirve para indicar al navegador que empieza el encabezado del documento; la etiqueta *<title>* especifica el texto que irá dentro de la barra de título del navegador, mientras que *<img>* sirve para hacer referencia a una imagen que está dentro del Web.

1.6.2 Ventajas y limitaciones del HTML

La mayor ventaja del HTML es que se permite desplazar rápidamente entre los sitios Web, a través de ligas o hipervínculos y en pocos minutos visitar varias decenas o cientos de páginas, por ejemplo, esta es una liga: *http://www.disney.com*

Las ligas se identifican, porque cuando se posiciona el puntero del mouse sobre ellas, el puntero se cambia a una curiosa manita. Sin embargo, el HTML está limitado porque las páginas no son interactivas, es decir no pueden procesar entrada de datos por el usuario, ni funcionar como si fueran aplicaciones de computadora; esto quiere decir que el funcionamiento se limita al esquema de la figura 1.2

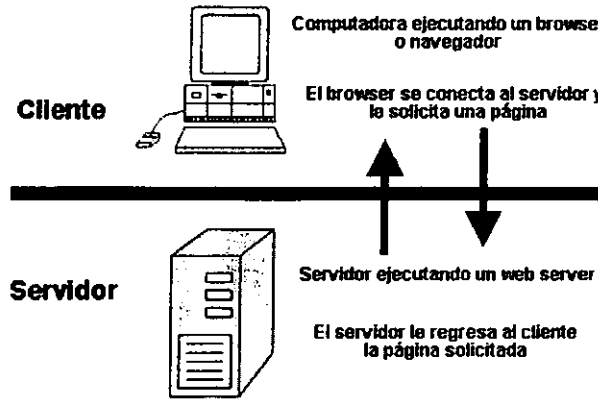


Figura 1.2 Funcionamiento del html.

Para lograr que el servidor tuviera menos carga de procesamiento, y el tiempo de respuesta a través de una página Web disminuyera, se inventaron tecnologías dentro de los navegadores, para que éstos mismos procesaran datos en la misma computadora del cliente, además teniendo en cuenta que debería existir compatibilidad entre las diversas plataformas de computadoras (UNIX, Windows 95, 98 ó NT, MacOS, OS/2 etc.).

1.6.3 Tecnología Java y ActiveX

La compañía Sun. Microsystems desarrolló una solución: Java. Que es un lenguaje de programación portable, robusto, seguro, orientado a objetos y de alto desempeño, cuya principal característica es que al compilarlo, en vez de generarse código ejecutable para una plataforma específica se genera un código de bytes (*bytecode*[2]), que es un código independiente y capaz de ejecutarse en cualquier computadora. Uno de los principales programas que se pueden hacer con esta tecnología, son los llamados Applets de Java(que son *bytecodes* especiales que se envían a través de la red, incrustados dentro de una página HTML), una vez compilados, son interpretados línea por línea por la Máquina Virtual de Java (JVM), que está implementada en el navegador (Brow*ser*) de Internet (Internet Explorer, o Netscape Navigator) Con esta tecnología un usuario puede ejecutar cualquier aplicación en cualquier computadora, siempre y cuando su navegador soporte Java. Actualmente los dos principales navegadores pueden correr Applets de Java [13].

Por otro lado, la contraparte de Microsoft, es la tecnología ActiveX que está basada en componentes de software, que pueden ser escritos en cualquier lenguaje y que, al igual que los Applets, se pueden incrustar dentro de páginas HTML [13].

Por ejemplo, si necesitas proporcionar la capacidad de análisis y calculo que Excel posee, a las aplicaciones de Visual Basic o dar formato a un documento con las herramientas de Microsoft Word o almacenar y administrar datos mediante el motor de bases de datos Microsoft Jet, la tecnología ActiveX permite realizarlo.

Todo esto se puede conseguir si construye aplicaciones con componentes ActiveX. Un componente ActiveX es un fragmento reutilizable de código y datos, compuesto por uno o más objetos creados con tecnología ActiveX. Las aplicaciones pueden utilizar componentes existentes, como los incluidos en aplicaciones de Microsoft Office, componentes de código, documentos ActiveX o controles ActiveX, antes llamados controles OLE.

Sin embargo, ActiveX, tiene la limitante que suele ser específico para la plataforma Windows, y sólo trabaja mediante Microsoft Internet Explorer.

[2] Bytecode: término en inglés para definir a un formado de código de bytes que son el resultado de compilar programas escritos en Java y que mediante la Máquina Virtual de Java, pueden correr en diferentes plataformas.

Resumen.

Los sistemas expertos fueron desarrollados por la comunidad de Inteligencia Artificial (IA), a mediados de los años 60; de todas sus áreas de investigación, ésta es una de las más aplicadas.

Los sistemas expertos, son programas de computadora que tratan de imitar los procesos de razonamiento y conocimiento que los expertos humanos hacen para la solución de un tipo específico de problemas.

Los sistemas expertos se componen de dos partes principales: un Ambiente de desarrollo y uno de consulta.

La Web, es uno de los múltiples servicios que ofrece Internet, debido a su facilidad de uso, su interfaz gráfica y su relativa interactividad se ha convertido en el servicio preferido de usuarios en todo el mundo.

El servicio Web se basa en la transmisión de hipertexto a través de la red y el cual es interpretado por los browsers o navegadores de Internet.

Con las herramientas adecuadas, los sistemas expertos pueden ser consultados en la Web.

Capítulo II

Descripción del problema y recursos necesarios.

Introducción.

Con el avance de Internet y la tecnología de comunicaciones, y junto con el desarrollo de la World Wide Web, se ha incrementado el desarrollo de nuevas herramientas para el aprovechamiento al máximo de todos los recursos computacionales basados en Internet y la Web, lo cual demanda una constante investigación para el desarrollo de nuevas tecnologías que permitan el mejor aprovechamiento de este recurso.

El presente capítulo da una descripción general del problema a resolver, los elementos necesarios para lograr el objetivo, así como los recursos que serán utilizados, tanto en hardware como software para el desarrollo e implementación del sistema como solución al problema que se plantea.

2.1 Planteamiento del problema.

En los últimos años, el uso de las computadoras para la solución de problemas específicos se ha incrementado, a tal punto, que se ha pasado de problemas específicos a cualquier tipo de problemas, es decir, todo se quiere resolver mediante la computadora, además de que éstas han demostrado ser herramientas útiles al ser humano para un desempeño más eficiente de su trabajo.

Cuando nos enfrentamos a un problema y la necesidad de llegar a una solución para este, es necesario:

Primero. Conocer el tipo de problema y su relación con el ambiente que esta en su entorno.

Segundo. Relacionarlo con otros problemas del mismo tipo,

Tercero. Buscar una estrategia de solución y lograr la solución de dicho problema.

Para esto, muchas veces es necesario recurrir a la ayuda de una persona especialista o especializada en el área de investigación relacionada con dicho problema. Ahora bien, qué sucede cuándo esta persona especialista no se encuentra disponible, ó es difícil de contactar; en estos casos es para los cuales se ha desarrollado una disciplina de un área de la computación conocida como Inteligencia Artificial para la solución de problemas de manera similar a lo que lo haría un especialista; esta disciplina ha dado como resultado la implementación y desarrollo de programas inteligentes que se conocen como Sistemas Expertos (SE).

Un Sistema Experto técnicamente hablando, es un programa de computadora que emula la aplicación del conocimiento que los expertos humanos hacen a fin de dar o aproximar las respuestas a la solución de problemas, para que estas sean lo más parecidas o similares a las que proporcionaría un experto humano [1].

Para construir este programa de computadora, es necesario extraer toda la experiencia y conocimiento del experto humano y hacer una representación de esta de una manera que sea fácil de procesar por la computadora; para ello se utilizan técnicas de representación del conocimiento y a la disciplina que se encarga de esta tarea se le conoce como Ingeniería del Conocimiento y a las personas que la aplican, se conocen como Ingenieros del Conocimiento [2].

Un ingeniero del conocimiento estudia la experiencia y conocimientos del experto humano, clasifica cada parte y al final tiene una recopilación de toda esa experiencia y conocimiento para la solución de problemas, la cual se implementa como una parte de un programa de computadora que resolverá el mismo tipo de problemas que el experto humano; aplicará los mismos criterios y experiencia para así obtener soluciones similares pero de una forma más rápida y económica.

Una vez que se ha terminado un sistema experto, éste estará listo para trabajar y distribuirse en lugares donde sea necesario; una manera de que éste sistema sea distribuido, es mediante la instalación autorizada y legal del sistema en varios lugares (computadoras) donde los usuarios del mismo podrán hacer consultas y obtener resultados del sistema. Esto proporciona un gran beneficio ya que como se mencionó antes, el sistema puede trabajar de manera independiente en varias computadoras.

El propósito de este trabajo, es determinar y obtener los elementos necesarios para el desarrollo de sistemas expertos y ponerlos a disposición de los usuarios a través de la Web, tomando en cuenta estos puntos, el sistema debe en forma general tener la capacidad de realizar las siguientes funciones:

- Contar con un ambiente de desarrollo para Sistemas Expertos (que serán programas de computadora) para la solución de problemas de áreas especificas.
- Probar y depurar los sistemas expertos desarrollados a fin de reducir los errores de operación.
- Contar con los elementos necesarios para montar dichos sistemas expertos a la disposición de usuarios a través de la World Wide Web, mediante técnicas de programación en la Web,
- Un medio físico de comunicaciones, que nos permita el enlace hacia la Web.

A continuación se detallan cada una de estas tareas.

2.2 Ambiente de desarrollo de Sistemas Expertos.

El ambiente de desarrollo, es la parte del sistema que nos permitirá capturar y representar el conocimiento del experto humano, esto se hace mediante técnicas específicas de representación del conocimiento.

Para esta parte, será necesario un ambiente de desarrollo el cual consiste en una herramienta para programación de sistemas expertos, esto es, una interfaz que permita llevar a cabo las diferentes etapas del desarrollo de un sistema experto, las cuales son: *construcción de un prototipo, formalización implantación, evaluación y evolución a largo plazo.* Existen herramientas comerciales, que pueden ser buscadas en diferentes partes y proveedores; estas herramientas de desarrollo son conocidas como shells (o cascarones) para desarrollo de sistemas expertos, las cuales proporcionan un gran avance en el desarrollo de un sistema experto ya que ofrecen todos los elementos (a excepción de la Base de Conocimiento) encapsulados en un solo programa para cumplir con las diferentes etapas de desarrollo, y todo en un solo sistema.

Así, diferentes shells comerciales pueden ser estudiados para determinar las características de operación que deberá tener el sistema.

Dado que nuestra meta es implementar una herramienta de desarrollo propia, esto debido a que la mayoría de los shells[1] comerciales al ser de procedencia extranjera, se encuentran en un idioma diferente al español, es necesario construir una herramienta de desarrollo propia que se ajuste a las necesidades descritas anteriormente; este sistema será diseñado e implementado en el desarrollo del proyecto (capítulo IV), y además contará con las siguientes ventajas:

- Tendrá todas las características y cualidades de shells comerciales evaluados.
- Tendrá cualidades propias que lo harán diferente a los comerciales evaluados y que serán más adecuadas a las necesidades y requerimientos del sistema en general.
- Podrá ser modificado o agregarle más funciones según el avance mismo del desarrollo de los sistemas.

El ambiente de desarrollo contará con técnicas de representación de conocimiento basado en reglas, esto nos lleva a concluir inmediatamente que los sistemas expertos que se desarrollen en este ambiente, serán también sistemas expertos basados en reglas. La técnica de representación de conocimiento mediante reglas es una de las más utilizadas en el desarrollo de sistemas expertos, esta se describe más detalladamente en el capítulo III.

[1] Shell: es un término utilizado en inglés para definir o programa para desarrollo de sistemas expertos.

2.3 Prueba y Depuración.

Una vez que se ha terminado de capturar el conocimiento y experiencia del experto humano, el siguiente paso es probar dicha base de conocimiento, las pruebas se llevan a cabo mediante la aplicación de procesos de inferencia (o razonamiento) a la base de conocimiento, este procesamiento lo realiza otra parte del sistema experto que se conoce como máquina de inferencia [2].

Una máquina de inferencia es la parte del sistema experto que trata de emular el razonamiento humano, y para ello utiliza técnicas y procedimientos de inferencia. Cuando a un experto humano se le presenta un problema él hace un razonamiento mediante el análisis por un lado del problema en sí, y por otro lado, mediante el análisis de su conocimiento y experiencia para la solución de este tipo de problemas; con base en esta técnica de análisis, el experto humano infiere una o varias soluciones para así llegar a una solución general.

De igual manera, la función de la máquina de inferencia es emular ese razonamiento. Una máquina de inferencia trabaja sobre la base de conocimiento, al realizar un análisis de ésta, y con base en los datos que ésta contiene mediante este proceso de razonamiento, genera nuevo conocimiento (nuevos datos) para así obtener soluciones similares a las que daría un experto humano.

Con esto, podemos deducir inmediatamente que nuestro sistema además de contar con un ambiente de desarrollo, también deberá contemplar una máquina de inferencia para prueba y depuración de los sistemas expertos desarrollados.

Para llevar a cabo su función, la Máquina de inferencia usa técnicas de razonamiento que se basan principalmente en una las siguientes: **Encadenamiento hacia adelante**, y **Encadenamiento hacia atrás** [2]; estas técnicas de razonamiento se detallan en el capítulo III.

Los shells comerciales generalmente disponen de una máquina de inferencia integrada, la cual aplica una o ambas técnicas de inferencia; además también cuentan con una interfaz de usuario que sirve para la comunicación directa entre el sistema y usuario.

Una vez que se termina de capturar la base de conocimiento, se hace correr la máquina de inferencia, la cual se ejecuta sobre la base de conocimiento y proporciona resultados, que son analizados junto con el experto humano, estos resultados preliminares son de gran importancia ya que nos permiten:

- Saber si el sistema experto en desarrollo va por el camino correcto.
- Determinar si a la base de conocimiento le faltan reglas, variables o parámetros que no se han tomado en cuenta.
- Hacer una comparación con los resultados que daría el experto humano.

- Detectar y corregir problemas de operación y ejecución, a fin de que estos no se presenten al momento de la operación real.

En cuanto al desarrollo propio, no se debe omitir esta parte por ningún motivo, es requisito indispensable que cuente con la máquina de inferencia para realizar pruebas y depuración de los sistemas expertos desarrollados y que esta permita aplicar los dos procesos de razonamiento Encadenamiento hacia adelante, y Encadenamiento hacia atrás. Como se menciono anteriormente, esto representa grandes ventajas sobre los shells comerciales, ya que permitirá, que posteriormente, se puedan integrar más técnicas y/o procesos de inferencia con el fin de hacerlo más eficiente y dinámico.

La función de esta parte del sistema se representa el diagrama de la figura 2.1.

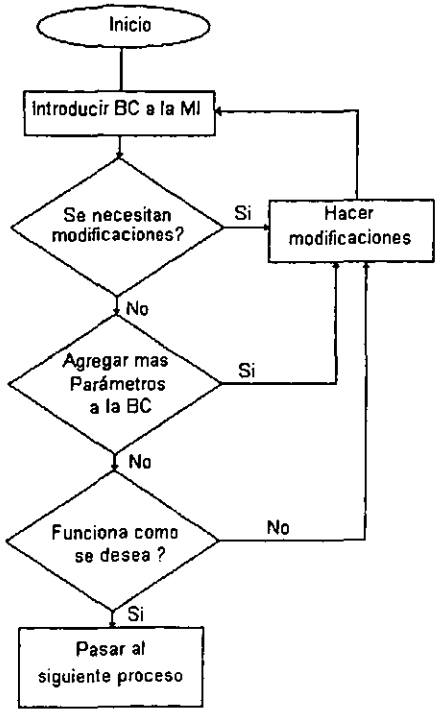


Figura 2.1 Función de la máquina de inferencia.

2.4 Distribución de los sistemas expertos a través de la Web.

Una vez terminada y depurada la base de conocimiento, ésta se debe integrar junto con la máquina de inferencia y la interfaz de usuario para obtener el producto final, un sistema experto capaz de trabajar en la Web.

Para llevar a cabo la distribución del sistema experto y esté disponible a los usuarios, esto se llevará a cabo a través de Internet, mediante técnicas de programación en Web.

Existen varias opciones que se pueden tomar en cuenta para realizar esta parte del sistema, la más común es mediante páginas html que permiten la inserción de pequeños programas independientes que se ejecutan dentro de dicha página; estos programas bien pueden ser sistemas expertos completos e integrados con su máquina de inferencia, base de conocimiento, e interfaz de usuario y que además se ejecutan dentro de una página html.

A estos programas que corren de manera independiente dentro de páginas html, se conocen como "Applets" [6], se hablará más al respecto de los Applets en el capítulo III.

Nuestro sistema de desarrollo propio, deberá contar con los elementos necesarios para montar en la Web los sistemas expertos desarrollados.

2.5 Medio físico de comunicación.

Al inicio de este capítulo se comentó que nuestra arquitectura está formada por dos elementos principales: el hardware y el software, estos elementos siempre son dependientes uno del otro.

Se ha estado hablando de los programas que se necesitan para la implementación del sistema, pero para que se pueda desarrollarlos, se necesitan las herramientas físicas que nos permitirán realizar esta tarea.

El medio físico de comunicaciones, consiste en un minired de computadoras conectada a Internet, la cual nos permitirá ejecutar tanto los programas de lenguajes de programación, así como el sistema final para el desarrollo y distribución de los sistemas expertos a través de la Web.

La red de computadoras servirá para dos tareas: al final correrá el sistema de desarrollo y distribución de los sistemas expertos, y además, servirá para desarrollar este sistema.

Así de esta manera, queda planteado el problema a resolver, debemos determinar e integrar cada uno de los elementos descritos anteriormente a nuestro sistema para que al final, podamos cumplir con los objetivos planteados, a manera general, la función del sistema se presenta en la figura 2.2 que se muestra a continuación.

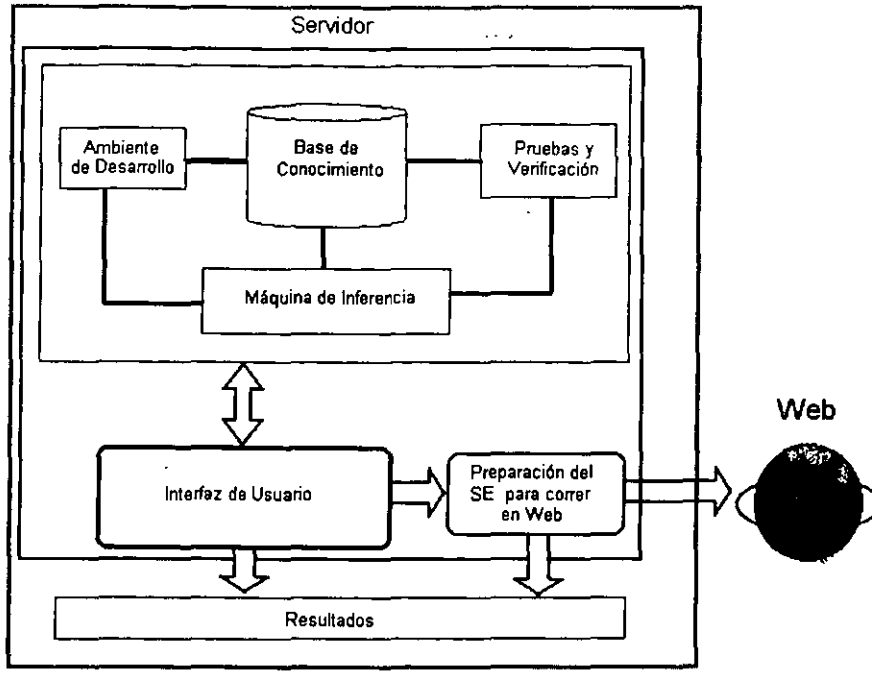


Figura 2.2 Servidor Web para Sistemas Expertos.

El servidor Web conectado a Internet, junto con la red de computadoras conforman y los sistemas de desarrollo, conforman la arquitectura necesaria para, el desarrollo e implementación de los sistemas expertos y su disposición a usuarios de todo el mundo a través de la Web.

Con esta arquitectura, podremos desarrollar y probar los programas antes de ser preparados para correr en la Web. Una vez que este listo, la forma de operación será como se presenta en la figura 2.3.

Un cliente que desee utilizar un sistema experto en alguna área específica de aplicación, podrá consultar las opciones disponibles de nuestro servidor y en un momento dado proporcionar sugerencias para el desarrollo de nuevos sistemas expertos en otros dominios.

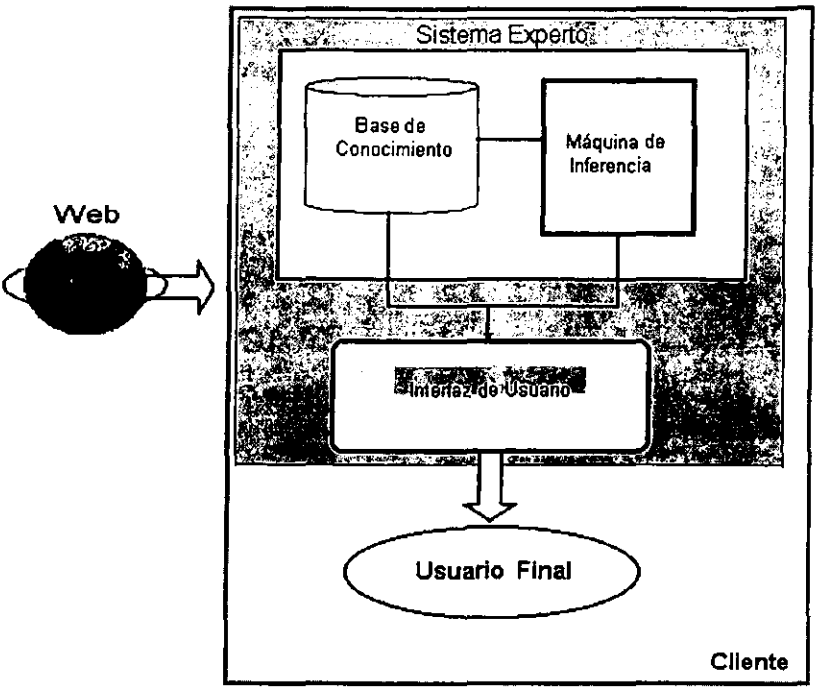


Figura 2.3 Uso de un Sistema Experto a través de la Web.

Resumen.

La arquitectura propuesta deberá contar con un ambiente de desarrollo para sistemas expertos, esto comprende la captura y representación del conocimiento, una máquina de inferencia para realizar las pruebas y depuración, por último dar un formato especial a los sistemas expertos para que estén a disposición de los usuarios a través de la Web.

El ambiente de desarrollo contará con técnicas de representación de conocimiento basado en reglas, ésta es una de las más utilizadas en el desarrollo de sistemas expertos por sus amplias facilidades de implementación.

El ambiente de desarrollo, cuenta con una máquina de inferencia que usa técnicas de razonamiento basadas en Encadenamiento hacia adelante, y Encadenamiento hacia atrás. Esta máquina de inferencia, se ejecuta sobre la base de conocimiento para proporcionar resultados, que son analizados junto con el experto humano.

Capítulo III.

Inteligencia Artificial y Sistemas Expertos.

Introducción.

El presente capítulo, explica de manera general el marco teórico de Inteligencia Artificial y los Sistemas Expertos. Se describen muchos de los conceptos que serán necesarios para el desarrollo e implementación del sistema, objeto de este trabajo (capítulo IV), entre ellos están: Base de Conocimiento, Representación de Conocimiento, Máquina de Inferencia, Reglas, Procesos de Razonamiento etc.

Se describen además varias propiedades de los sistemas expertos basados en reglas, que es también la base del desarrollo e implementación del sistema basado en Web.

3.1 Inteligencia Artificial.

La tecnología de los sistemas expertos se deriva de una disciplina de investigación, conocida como Inteligencia Artificial, que es una rama de las ciencias de la computación relacionada con el diseño e implementación de programas de computadora que son, en cierta medida capaces de emular capacidades humanas para: la solución de problemas, la percepción visual y entendimiento del lenguaje; esta tecnología ha sido aplicada exitosamente a muchos dominios como la química orgánica, exploración minera y la medicina interna entre otros [1].

La inteligencia artificial es un campo de estudio de las ciencias de la computación que tiene como meta hacer que la computadora razone de manera similar a como lo hace un experto humano [5]. Desde un punto de vista más práctico, la meta principal de la inteligencia artificial, es hacer que las computadoras sean más útiles para los humanos, esto puede ser logrado produciendo programas de computadoras más fáciles de usar con interfaces en lenguaje natural. Una segunda meta y de igual importancia, es para entender mejor la inteligencia humana.

3.2 Sistemas Expertos.

Un Sistema Experto, es un programa de computadora en el que se representa el conocimiento y experiencia de un especialista humano y además emula el razonamiento que éste aplica a la solución de problemas, para dar una o más recomendaciones. Tales programas están diseñados para que realicen las mismas funciones de la expertice humana. Los sistemas expertos mimetizan el comportamiento de los expertos humanos, éstos usan información que el usuario les suministra para obtener una opinión sobre cierto objeto. Así el sistema experto

hace una serie de preguntas hasta que puede identificar un objeto que se relaciona con las respuestas del usuario.
Las tareas típicas de un sistema experto envuelven a la interpretación de datos, diagnostico de fallas, análisis estructural de objetos complejos (como compuestos químicos), configuración de objetos complejos (como sistemas de computo), y planear secuencias de acción.

Los sistemas expertos existen principalmente por dos razones: primero, son programas generalmente útiles y prácticos que satisfacen una necesidad; y en segundo, son alcanzables para muchos usuarios. Estas son las razones por las cuales ellos constituyen uno de los mayores éxitos comerciales de la inteligencia artificial.

Los sistemas expertos como una entidad especializada de la inteligencia artificial pueden dar recomendaciones efectivas a un individuo u organización a nivel de toma de decisiones. Este propósito especial de los sistemas de computación, provee individuos y organizaciones con un alto nivel de expertise, conocimiento explícito y accesible y también consultas permanentes en un dominio específico de aplicación. Explorando la potencialidad de la tecnología de Internet para soportar el acceso a expertos remotos, representa nuestra principal meta. La estructura y servicios de Internet pueden ser usados para proveer el acceso a estos sistemas expertos y esto puede ser un proceso efectivo para reducir significativamente la inversión inicial y esfuerzos en la instalación y distribución de sistemas expertos remotos.

3.2.1 Arquitectura de los sistemas expertos.

Los sistemas expertos emplean una amplia variedad de arquitecturas específicas, principalmente porque una arquitectura es más aplicable que otra cuando se considera una aplicación dada. Actualmente se lleva a cabo una vasta investigación para estudiar diferentes aspectos de las arquitecturas para sistemas expertos y subsiste un considerable debate al respecto.

A pesar de las diferencias significativas, la mayoría de las arquitecturas tienen muchos componentes en común [2]. La figura 3.1 muestra una arquitectura general con los componentes típicos:

El usuario de un sistema experto puede estar operando en cualquiera de los siguientes modos:

- *Verificador.* El usuario intenta comprobar la validez del desempeño del sistema.
- *Tutor.* El usuario da información adicional al sistema o modifica el conocimiento que ya está presente en el sistema.
- *Alumno.* El usuario busca rápidamente desarrollar pericia personal relacionada con el área específica mediante la recuperación de conocimientos organizados y condensados del sistema.

- *Cliente.* El usuario aplica la pericia del sistema a tareas específicas reales.

El reconocimiento de las características anteriores contrasta con la percepción de un simple papel en los sistemas tradicionales de software.

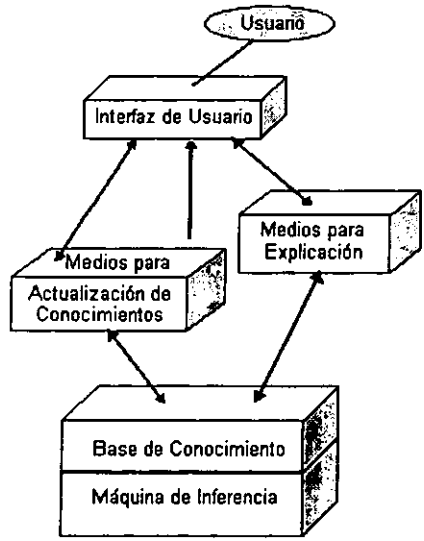


Figura 3.1 Arquitectura típica de un Sistema Experto

Facilidades de interfaz con el usuario. Deben aceptar información del usuario y traducirla en una forma aceptable para el resto del sistema o aceptar información proveniente del sistema y convertirla en una que el usuario pueda entender.

Idealmente, esta facilidad se compone de un sistema procesador de lenguaje natural que acepta y devuelve esencialmente información en la misma forma como es aceptada u ofrecida por un experto humano.

Las facilidades de interfaz del usuario a menudo se diseñan para reconocer el modo en que el usuario esta operando, su nivel de pericia, y la naturaleza de la transacción. Aunque el dialogo en el lenguaje natural no es aún realizable, la comunicación con un sistema experto debe ser tan natural como sea posible, ya que el sistema trata de sustituir el desempeño humano.

Sistemas de almacenamiento y generación de conocimiento. El almacenamiento consta de una base de conocimiento y una máquina de inferencia. Es el corazón de un sistema experto. La función de este sistema consiste en almacenar confiablemente los conocimientos del experto, para recuperarlos e inferir nuevos conocimientos cuando se requiera.

Base de conocimiento. Representa un depósito de las primitivas del conocimiento (por ejemplo: hechos fundamentales, reglas de procedimiento y heurísticas) disponibles para el sistema, este conocimiento almacenado en la base de conocimiento, establece la capacidad del sistema para actuar como un experto.

En general, el conocimiento se almacena en forma de hechos y de reglas, pero los esquemas específicos empleados para almacenar la información varían grandemente. El diseño de este esquema de representación de conocimiento afecta el diseño de la máquina de inferencia, el proceso de actualización del conocimiento, el proceso de explicación y la eficiencia global del sistema. La selección del esquema de representación del conocimiento es una de las decisiones más críticas en el diseño de un sistema experto.

3.2.2 Base de Conocimiento.

La Base de Conocimiento (BC), es en esencia el alma del sistema experto, ya que es el elemento principal que determina el comportamiento de sistema experto; una base de conocimiento es la representación tanto de la experiencia, el conocimiento y la forma de aplicación de este para la solución de problemas que el experto humano posee; una BC es la expertise o experiencia que hace que un sistema experto emule a un experto humano [2].

3.2.2.1 Ingeniería de conocimiento.

El proceso mediante el cual se construye una Base de Conocimiento es conocido como Ingeniería del conocimiento. Un ingeniero de conocimiento es aquel que realiza investigaciones en un dominio en particular y crea una representación formal de los objetos y relaciones del dominio, es frecuente que los ingenieros de conocimiento estén capacitados en la representación aunque no sean expertos en el dominio que manejan. Normalmente el ingeniero de conocimiento entrevista a los verdaderos expertos de un dominio particular, para que estos le enseñen lo necesario sobre el dominio de interés y le deslinden las fronteras del conocimiento respectivo, por medio de un proceso denominado **Adquisición del Conocimiento** [9]. Esto se realiza antes de crear el proceso de las representaciones formales, o simultáneamente con éste.

No basta con estudiar la sintaxis y la semántica de un lenguaje de representación para convertirse en un diestro ingeniero del conocimiento. Antes de poder lograr un buen sitio con cualquier lenguaje, es necesaria mucha practica y el estudio de una gran cantidad de ejemplos, ya sea de un lenguaje de programación, razonamiento o de comunicación.

La ingeniería de conocimiento es el proceso de adquirir el conocimiento del área específica y estructurarlo en la base de conocimiento, la figura 3.2, muestra tal y como se da este proceso.

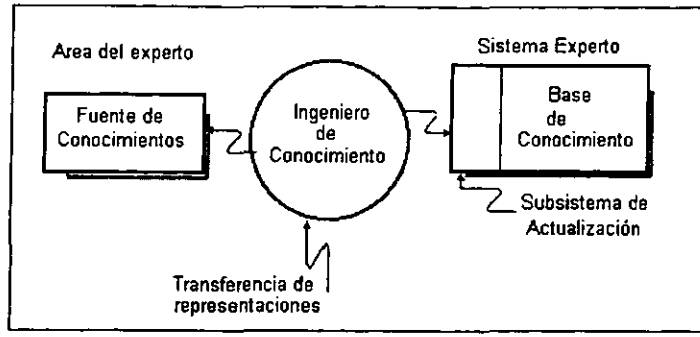


Figura 3.2 Adquisición del conocimiento.

Aunque los conocimientos pueden conseguirse de una gran variedad de fuentes incluyendo la documentación y los sistemas de información computacional existentes, la mayor parte de él, debe obtenerse de personas expertas en el área específica.

Un ingeniero de conocimiento obtiene los conocimientos del área del experto y los transporta a la base de conocimiento de acuerdo con las normas de representación del sistema. La función de adquisición de conocimiento es comúnmente, el aspecto de mayor dificultad en la construcción de un sistema experto. Esto se debe principalmente al hecho de que el proceso requiere comunicaciones humanas ampliadas, entre el ingeniero de conocimiento y el experto del área.

3.2.2.2 Representación del conocimiento (Generalización y Abstracción).

La generalización y abstracción, pueden ser usadas exitosamente en la representación lógica y semántica del conocimiento [9]. Por ejemplo, sí sabemos que todas las compañías tienen presidente y que todas las casas de bolsa son compañías, entonces podemos inferir y generalizar que las casas de bolsa tienen presidentes.

De manera similar si la computadora sabe que en una cierta compañía todos los ingenieros tienen un sueldo mensual, y que los contadores y los analistas de sistemas también tienen un sueldo mensual, eventualmente la computadora puede (mediante un proceso de inferencia establecido) concluir que todos los profesionales que trabajan en una compañía, tienen un sueldo mensual.

Uno de nuestros objetivos principales es usar el conocimiento para hacer inferencias, pero cómo es que se utiliza para: contestar preguntas, razonar o sacar conclusiones. Para ello utilizamos reglas de inferencia para manipular expresiones lógicas, y crear nuevas expresiones.

Existen muchas reglas básicas de razonamiento que permiten la manipulación de expresiones lógicas para crear nuevas expresiones, la más importante de estas reglas es la llamada **Modus Ponens** [1] [2] [9].

Modus Ponens, de acuerdo con este procedimiento, si tenemos una regla determinada "Si A entonces B" y sabemos que A es verdadera, entonces es válido concluir que B también es verdadera, en términos de lógica esto se expresa como:

$$[A \text{ y } (A \rightarrow B)] \rightarrow B$$

Nota:(A→B se lee A implica B)

Donde A y (A→B), (A→B se lee A implica B) son proposiciones en una base de conocimiento, dada esta expresión podemos reemplazar ambas proposiciones con la proposición B. En otras palabras podemos usar Modus Ponens para sacar la conclusión que B es verdadera si las primeras dos expresiones lo son. A continuación, tenemos un ejemplo más claro:

A: Esta soleado.
B: Vamos a la playa.
A→B: **Si** esta soleado **Entonces** vamos a la playa.

La primera premisa simplemente establece que es un día soleado. La segunda premisa dice que iremos a la playa además si A, y A implica B, son verdaderas, B es verdadera. Usando Modus Ponens puedes entonces deducir que iremos a la playa.

3.2.2.3 Procesos de razonamiento.

Una vez que la base de conocimiento ha sido completada, esta lista para ser usada, para hacer esto necesitamos otro programa de computadora que nos permita el acceso al conocimiento, con el fin de hacer inferencias y decisiones para la solución del problema. Este programa, es un algoritmo que controla a algún proceso de razonamiento y usualmente se le conoce como **Máquina de Inferencia (MI)** o programa de control [1] [2] [8] [9]. En sistemas basados en reglas es también conocido como **Interpretador de Reglas (IR)**.

El programa controlador dirige una búsqueda a través de la base de conocimiento, este proceso puede involucrar la aplicación de reglas de inferencia en lo que llamamos correspondencia de patrones (en inglés Patern matching); este programa decide qué regla analizar, qué alternativa eliminar y qué atributo comparar. Los programas de control más populares son los que se basan en dos procesos de inferencia conocidos como: *Encadenamiento hacia Adelante y Encadenamiento hacia atrás.*

Hay muchas formas en que la gente razona y resuelve problemas, esto se traduce en métodos específicos de inferencia o razonamiento, por mencionar algunos tenemos:

- *Razonamiento Deductivo.*- Es un proceso en el cual las premisas generales son usadas para obtener una inferencia específica. Este razonamiento se mueve de un principio general a una conclusión específica.

- *Razonamiento Inductivo.*- Usa un número establecido de hechos o premisas para obtener alguna conclusión general, esto se ve más claro en el siguiente ejemplo:

  Premisa: La falla de diodos causa que un equipo electrónico falle.
  Premisa: La falla de transistores causa que un equipo electrónico falle.
  Premisa: La falla de circuitos integrados causa que un equipo electrónico falle.

  Conclusión: Los semiconductores defectuosos son una causa que los equipos electrónicos fallen.

- *Razonamiento Formal.*- envuelve una manipulación sintáctica de estructuras de datos para deducir nuevos hechos, siguiendo reglas prescritas de inferencia. Un ejemplo típico, es la lógica matemática usada para probar teoremas en geometría.

- *Razonamiento Numérico.*- usa los modelos matemáticos o de simulación para resolver problemas.

3.3 Procesos de Inferencia.

Una vez que ya tenemos un lenguaje razonable para representar el conocimiento y también, contamos con una regla razonable de inferencia (por ejemplo Modus Ponens) que nos permita utilizar dicho conocimiento debemos construir un programa de razonamiento.

La regla Modus Ponens generalizada se puede utilizar de dos formas. Podemos empezar por las oraciones que están en la base de conocimiento y generar nuevas conclusiones, de las que, a su vez, se pueden obtener nuevas inferencias. A lo anterior se le conoce como **Encadenamiento hacia Adelante**. Se utiliza por lo general cuando se incorpora en la base de datos un nuevo hecho y deseamos generar sus consecuencias. O bien, podemos empezar por algo que deseamos demostrar y se buscan las oraciones que nos permitirán llegar a tal conclusión y por último, tratar de establecer las premisas correspondientes. A esto segundo, se le conoce como **Encadenamiento hacia Atrás** puesto que utiliza el Modus Ponens de manera inversa, este proceso, se utiliza por lo general cuando es necesario demostrar una meta específica.

La inferencia con reglas, implica la implementación del Modus Ponens; el mecanismo de inferencia en muchos sistemas expertos comerciales usa esta regla de inferencia, la cual se refleja en un mecanismo de búsqueda con la máquina de inferencia. La prueba de la premisa o conclusión de la regla puede ser tan simple como una correspondencia de patrones simbólicos en la regla con un patrón similar en la aserción.

Cada una de las reglas puede ser checada para ver si sus premisas o conclusión, pueden ser satisfechas mediante aserciones hechas previamente, y este proceso puede hacerse en una de las dos direcciones hacia adelante o hacia atrás, y se continuará hasta que ya no mas reglas puedan ser disparadas, o cuando la meta haya sido alcanzada.

3.3.1 Encadenamiento hacia Adelante.

Por lo general el Encadenamiento hacia Adelante se activa por la incorporación de un nuevo hecho *p* a la base de conocimiento. Por ejemplo, se puede incorporar como parte del proceso, la idea consiste en determinar todas las implicaciones cuya premisa sea *p*, y luego, si ya se sabe que las otras premisas son válidas, podemos añadir la consecuencia de la implicación a la base de conocimiento con lo que se activa así la obtención de más inferencias.

El Encadenamiento hacia Adelante es una aproximación de un manejador de datos [8]. En este proceso empezamos con toda la información disponible tal y como esta o bien de una idea básica y entonces tratamos de sacar conclusiones la figura 3.3 representa este proceso de inferencia.

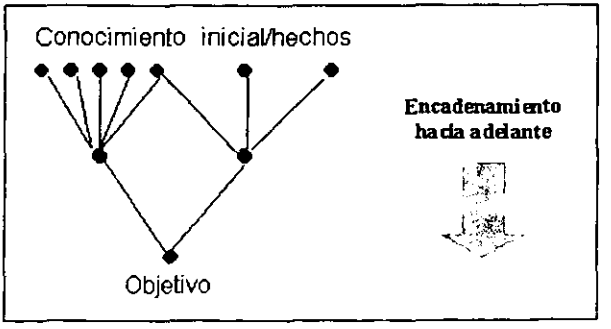


Figura 3.3 Encadenamiento hacia Adelante.

La computadora analiza el problema por medio de una búsqueda de los hechos que corresponden a la parte SI (IF) de una regla del tipo SI-Ent. por ejemplo, sí una cierta máquina no esta trabajando, la computadora checa el suministro eléctrico a dicha máquina. Como cada regla es probada, el programa trabaja de

esta forma hacia la obtención de una conclusión. Con el fin de entender mejor este proceso, tomemos como ejemplo el siguiente conjunto de reglas:

R1: Si A y C, Ent. E
R2: Si D y C, Ent. F
R3: Si B y E, Ent. F
R4: Si B Ent. C
R5: Si F Ent. G

y supongamos que nuestros datos iniciales o hechos son los siguientes:

A es verdadero, y B es verdadero.

El proceso de Encadenamiento hacia Adelante procede de la siguiente manera:

Inicio: como se sabe que A y B son verdaderos, estos son almacenados en una parte del sistema que se conoce como **Memoria de Trabajo (MT)** [1][7][8] el sistema experto comienza con uno de ellos (tomemos primero a A) y busca una regla que incluya a A en el lado Si. En nuestro caso, es R1, la cual tiene como conclusión a E.

Paso1: El sistema trata de verificar E. Como A es un hecho, es necesario probar a C con el fin de concluir a E. El sistema ahora trata de encontrar a C en los hechos. Como no está, el sistema busca una regla donde C este en la parte Ent. y esta es R4.

Paso2: El sistema prueba R4, C es verdadera, porque es comparada con B la cual se sabe que es verdadera, ya que esta en la memoria de trabajo como un hecho, luego entonces C es agregada a la memoria de trabajo.

Paso3: Ahora se dispara R1 y a E le es asignado el valor verdadero, y además E se agrega a la memoria de trabajo. Inmediatamente que un nuevo hecho o dato es inferido, a continuación se busca una regla que contenga dicho hecho en la parte SI, para nuestro caso el sistema encuentra que R3 tiene a E en su parte Si.

Paso4: Como B y E son verdaderos (están en MT), la regla R5 es disparada y F le es asignado el valor verdadero, y además F se agrega a la memoria de trabajo y nuevamente el sistema busca alguna regla que contenga a F en la parte Si, para nuestro caso el sistema encuentra que R5 tiene a F en su parte Si.

Paso5: Ahora R5 se dispara ya que F es verdadero y a G le es asignado el valor verdadero, y además G se agrega a la memoria de trabajo y nuevamente el sistema busca alguna regla que contenga a G en la parte Si, para nuestro caso ya no hay ninguna regla que cumpla con esta condición y el sistema se detiene.

Como habrá podido observar, el encadenamiento hacia adelante va formando un cuadro de la situación gradualmente, conforme van llegando nuevos datos. Sus

procedimientos de inferencia no están enfocados a resolver un problema en particular; por ello, se le conoce como procedimiento activado por datos o gobernado por datos. En ocasiones el encadenamiento hacia adelante genera muchas conclusiones irrelevantes, en estos casos es mejor recurrir al encadenamiento hacia atrás donde todos los esfuerzos se concentran en los datos que no interesan.

3.3.2 Encadenamiento hacia Atrás

El Encadenamiento hacia Atrás es un manejador de metas en el cual empezamos de una meta esperada o hipótesis, luego vemos las evidencias que soportan o contradicen dicha hipótesis. A veces, esto trae consigo la formulación y comprobación de hipótesis intermedias (o subhipótesis) [8] la figura 3.4 muestra este proceso de inferencia.

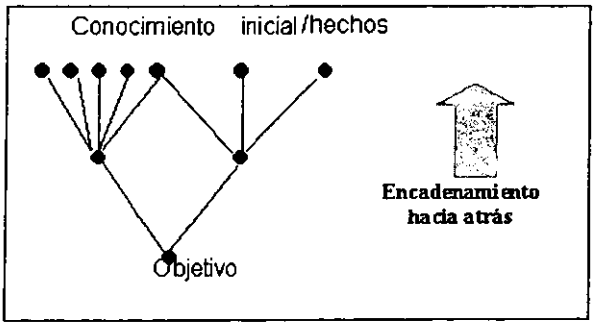


Figura 3.4 Encadenamiento hacia Atrás.

El encadenamiento hacia atrás está hecho para buscar todas las respuestas posibles a una pregunta formulada a la base de conocimiento. Por ello, este proceso de razonamiento cuenta con la funcionalidad requerida por el procedimiento PREGUNTAR. El algoritmo de encadenamiento hacia atrás lo que hace primero es verificar si las respuestas se pueden obtener directamente a partir de las oraciones de la base de conocimiento. Luego busca todas las implicaciones cuyas conclusiones respectivas unifican con la consulta y trata de establecer las cuales son las premisas de tales implicaciones, también mediante el encadenamiento hacia atrás. Si la premisa es una conjunción, entonces el encadenamiento hacia atrás la procesará elemento por elemento construyendo así el elemento unificador de toda la premisa para entender mejor, tomemos el siguiente ejemplo:

Consideremos las siguientes variables:

A = Tienes 10000 pesos.
B = Eres menor de 30 años.

C = Tu nivel de estudios es de licenciatura.
D = Tus ingresos anuales son de al menos 80000 pesos
E = Invertir en seguros.
F = Invertir en valores crecientes
G = Invertir en acciones de IBM.

Cada una de estas variables puede ser contestada como verdadera (Si) o falsa (No).

Los hechos: Asumamos que un inversionista tiene los 10000 pesos (es decir A es verdadero) y tiene una edad de 25 años (es decir B es verdadero). Él desea una recomendación acerca de invertir en acciones de IBM. ¿Se cumple o no la meta?.

Las reglas: Asumamos también que nuestra base de conocimiento tiene las siguientes reglas:

R1: **Si** tienes 10000 pesos **y** nivel de estudios de licenciatura, **Ent.** deberías invertir en seguros.
R2: **Si** los ingresos anuales son de al menos 80000 pesos **y** nivel de estudios de licenciatura, **Ent.** deberías invertir en valores crecientes
R3: **Si** eres menor de 30 años **y** invierte en seguros **Ent.** deberías invertir en valores crecientes.
R4: **Si** eres menor de 30 años **Ent** tiene grado de licenciatura.
R5: **Si** quieres invertir en valores crecientes **Ent.** las acciones recomendables son las de IBM.

Las reglas pueden ser escritas de la siguiente manera, a fin de simplificar el trabajo.

R1: **Si** A **Ent.** C.
R2: **Si** D **y** C **Ent.** F.
R3: **Si** B **y** E **Ent.** F.
R4: **Si** B **Ent.** C.
R5: **Si** F **Ent.** G.

Nuestra meta es determinar si invertimos o no en acciones de IBM, es decir, determinar si dadas las condiciones o hechos iniciales, G es verdadero.

Punto de inicio: En el proceso de encadenamiento hacia atrás, empezamos por buscar todas las reglas que tienen la meta (G) en la parte Ent, es decir, como conclusión. Como la regla R5 es la única que cumple con esta condición, el proceso comienza con estas reglas, si hubiera mas reglas con G como conclusión, la máquina de inferencia ejecuta un procedimiento para manejar dicha situación.

Paso1: Para probar si G es rechazada ó aceptada, lo que el sistema hace ahora es, analiza la memoria de trabajo para ver sí G está en ella. En este momento, lo único que hay en la memoria de trabajo, son los hechos:

A es verdadero.
B es verdadero.

Como no esta G, el sistema continua con el paso2

Paso2: R5 dice que si F (invertir en valores crecientes) es verdadera, entonces G (Invertir en acciones de IBM) también es verdadera. Si podemos concluir que la premisa de R5 sea verdadera o falsa, habremos resuelto el problema, pero no se sabe nada acerca de F si es verdadera o falsa, ¿qué es lo que se hace ahora?. Note que F además de ser la premisa de R5, también es la conclusión de R2 y R3, por lo tanto, para encontrar si F es verdadera debemos checar estas dos reglas.

Paso3: Probamos primero R3; si C y D son verdaderos, entonces F es verdadero. Pero tenemos un problema, D no es conclusión de ninguna regla y tampoco es un hecho. El sistema debe entonces invalidar dicha regla y buscar otra regla que tenga a D como conclusión, o bien, pedir al usuario una respuesta para el valor de D (ingreso anual de la persona).

Lo que el sistema hará depende de los procedimientos programados en la máquina de inferencia. Normalmente, se le pregunta al usuario por nueva información cuando ésta es desconocida o no puede ser inferida. Ahora bien, sí el usuario no proporciona información para determinar el valor de esta regla, abandonamos a R2 y continuamos con la siguiente regla, en este caso es R3, esta acción es conocida como retroceso o (backtracking). La máquina de inferencia debe estar programada para llevar a cabo los retrocesos.

Paso4: Nos vamos a R3, probamos B y E, sabemos que B es verdadero porque es un hecho en la MT, y para probar a E tenemos que analizar a R1 porque E es conclusión de esta regla.

Paso5: Examinamos a R1, es necesario probar si A y C son verdaderos. A es verdadero, porque es un hecho en la MT. Para probar a C, nos vamos a R4 de donde C es conclusión de esta regla.

Paso6: Examinamos la regla R4 para que C sea verdadero, debemos probar si B es verdadero, lo cual se cumple ya que B es un hecho en la MT, por lo tanto disparamos a R4 y C se le asigna el valor verdadero y es almacenado en la MT.

Paso7: Si C es verdadero, entonces se dispara R1 y a E se le asigna el valor verdadero y es almacenado en la MT. Como ahora E y B son verdaderos, se dispara R3 y a F se le asigna el valor verdadero y es almacenado en la MT. Ahora F ya es verdadero, por lo que se dispara R5 y a G se le asigna el valor verdadero y es almacenado en la MT. Dando como resultado que la meta buscada se ha logrado y la recomendación se convierte en G verdadero (Invertir en Acciones de IBM). Este procedimiento se muestra en la figura 3.5.

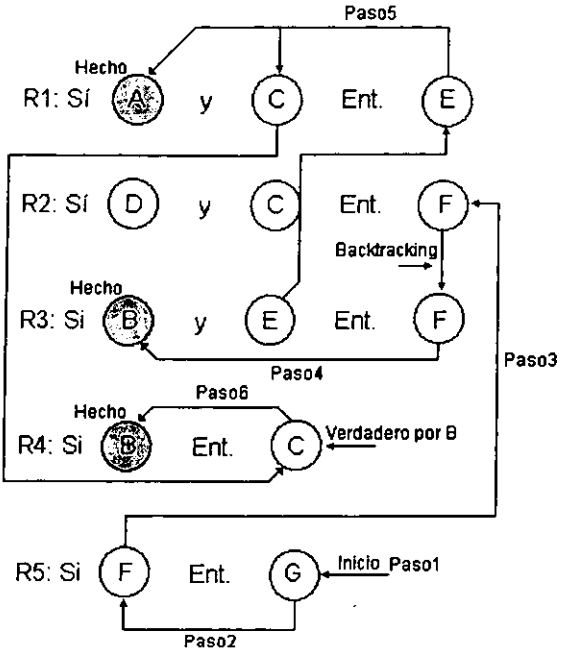


Figura 3.5 Búsqueda de la meta usando encadenamiento hacia atrás.

Como puede observar, durante el proceso de encadenamiento hacia atrás, el proceso de búsqueda se mueve de la parte Ent. a la parte Si, parte Ent. a la parte Si etc. Esta es una típica búsqueda de patrones en encadenamiento hacia atrás.

3.3.3 Máquina de Inferencia.

Los sistemas expertos deben por su naturaleza tratar en forma flexible con situaciones cambiantes. La capacidad para responder ante situaciones cambiantes depende de la habilidad que este tenga para inferir nuevos conocimientos a partir de los ya existentes. A manera de ejemplo sencillo, consideremos los dos hechos básicos siguientes:

Todos los animales respiran oxígeno.
Todos los perros son animales.

Se puede inferir un nuevo hecho, "todos los perros respiran oxígeno" a partir de los dos hechos anteriores. Para responder a una situación dada, un sistema experto debe aplicar el conocimiento apropiado, lo cual constituye un elemento clave del procesamiento de un sistema experto.

La máquina de inferencia es el sistema de software que ubica los conocimientos e infiere nuevos usando la base de conocimiento [2][5]. El paradigma de la máquina de inferencia es la estrategia de búsqueda que se emplea para producir el conocimiento demandado. Varios paradigmas diferentes se emplean en un sistema experto, pero la mayoría de ellos se basan en dos conceptos fundamentales: *Encadenamiento hacia atrás* que es un proceso de razonamiento descendente, que se inicia a partir de los objetivos deseados y trabaja hacia atrás en dirección las condiciones pre-requisito, y *encadenamiento hacia adelante* que es un procesamiento de razonamiento ascendente que se inicia con las condiciones conocidas y trabaja hacia adelante para alcanzar los objetivos deseados

En muchas áreas complejas, el conocimiento crece y cambia constantemente por lo cual la base de conocimientos se debe modificar en el mismo sentido. Este proceso puede tomar una de tres formas fundamentales según como se describe a continuación:

La primera forma es la *Actualización manual de conocimientos.* En este caso, la actualización se lleva cabo por un ingeniero de conocimiento, quien interpreta la información ofrecida por un experto del área y actualiza la base de conocimiento mediante el uso de un sistema limitado de actualización.

La segunda forma, que representa el estado del arte en un sistema experto, el experto en el área ingresa directamente el conocimiento revisado sin la mediación de un ingeniero de conocimiento. En este caso, el sistema de actualización debe ser más elaborado.

En la tercera forma, *el aprendizaje mecánico*, el sistema genera nuevos conocimientos en forma automática y se basa en generalizaciones deducidas de experiencias anteriores. El sistema, en efecto, aprende nominalmente de la experiencia e idealmente el mismo sistema se actualiza. Este proceso que aún está en estado conceptual, es tema de mucha investigación. La habilidad para aprender es un componente importante de la inteligencia y al ofrecer completamente esta potencialidad mejoraría las capacidades de un sistema experto.

3.4 Sistemas expertos basados en reglas.

Los sistemas basados en reglas son la representación más popular de los sistemas expertos, tales reglas son referidas como SI-ENT. o reglas de producción, también se usan por las siguientes razones [9]:

- La mayoría de los paquetes de sistemas expertos (Shell) emplean reglas.
- Son normalmente mucho mas baratos que los que emplean modos alternativos de representación, y no requieren hardware especial para su operación, pueden trabajar en simples Computadoras PC.

- El amplio campo de aplicación disponible en un sistema basado en reglas, permite al ingeniero de conocimiento enfocar su atención en la fase mas crítica del desarrollo de un sistema experto, esto es la base de conocimiento (BC).
- Las reglas representan un modo natural de representación del conocimiento, así el tiempo necesario para aprender a usarlas es mínimo.
- La curva de aprendizaje para un sistema basado en reglas, es mucho más escarpada que para los métodos alternativos.
- Las reglas son transparentes y ciertamente más transparentes que los modos de representación del conocimiento basados en reglas.
- Las reglas pueden ser fácilmente modificadas. En particular: agregar, borrar revisar y sus procesos son relativamente sencillos.
- Los sistemas basados en reglas pueden ser empleados para mimetizar muchas características de los esquemas de representación basados en frames.
- La validación del contenido de un sistema basado en reglas (es decir determinación de la integridad y consistencia de la representación) es un proceso relativamente simple. Para los Frames o redes neuronales, el proceso es normalmente difícil o imposible.

Asumiremos que nuestras reglas son deterministicas, esto es que no que no hay incertidumbre con las memorias o los hechos usados o las conclusiones alcanzadas.

Tales reglas son del tipo **SI-ENT.** aunque en algunos casos se extienden a **SI-ENT.-EOC** (EOC: En Otro Caso) por ejemplo tenemos las siguientes reglas:

R1: **SI** El estudiante tiene un puntaje de 1350 o más
ENT. Admitir al estudiante al programa de graduación.
EOC No admitir al estudiante.

Que es equivalente a tener las dos reglas siguientes:

R1: **SI** El estudiante tiene un puntaje de 1350 o más
ENT. Admitir al estudiante al programa de graduación.

R2: **SI** El estudiante tiene un puntaje menor a 1350
ENT. No admitir al estudiante.

Una designación alternativa para las reglas **SI-ENT.** es los enunciados Condición-Acción o Premisa-Conclusión. Notamos también, que las reglas de producción contienen (o implican) tripletas Objeto-Atributo-Valor (O-A-V), por ejemplo, para las reglas anteriores tenemos:

Objeto = Estudiante.
Atributo = Puntaje.
Valor = 1350 ó más alto ó más bajo.

Otra tripleta O-A-V es implicada en la parte ENT. o conclusión:

Objeto	=	Estudiante.
Atributo	=	Admisión estatus.
Valor	=	Si o No

Nótese que la primera tripleta, compara o prueba un valor de un atributo dado, mientras que en la conclusión se asigna un valor a otro atributo dado.
También puede haber reglas que tengan muchas premisas y muchas conclusiones unidas por algún conector lógico como ejemplo tenemos las siguientes reglas:

Regla A:	Si	La condición del cielo es despejado.
	Y	La temperatura es baja.
	Ent.	La probabilidad de día fresco es alta.
	Y	Las plantas de afuera deben ser cubiertas.
Regla B:	Si	La edad del auto es nueva.
	o	La condición del auto es buena.
	Ent.	El auto arranca.
	Y	El viaje será seguro.

Las cláusulas en las premisas, pueden ser conectadas por medio de Y (o conjunción) y O (o Disyunción), pero las conclusiones solo se deben hacer con Y, ya que todas las conclusiones deben ser verdaderas.

Propiedades del par Atributo-Valor: como se habrá notado, cada premisa y conclusión contienen atributos y valores que están asociados a un objeto por ejemplo

Si	El promedio es igual o excede 3.5
Ent.	Aceptar al estudiante en el cuadro de honor

En esta regla el atributo para la premisa es "promedio" que es comparado contra 3.5 o mayor, aunque el objeto no esta explícitamente indicado, sabemos que se trata de un estudiante.

El par Atributo-valor, es un bloque de construcción fundamental de la premisa o conclusión y luego entonces también de la regla de producción. Asociado a cada par Atributo-Valor, están una serie de propiedades que son:

- Nombre (del atributo).
- Tipo (La clase de los valores asociados con el atributo, simbólicos o numéricos).
- Valores legales (conjunto de valores aceptables).
- Valores específicos (o múltiples valores).
- Factores de confidencia (asociados con cada atributo).

También hay dos tipos de cláusulas: Premisas y Conclusiones, otras propiedades asociadas con estas son:

- Simple vs múltiple (o compound cláusulas).
- Conjuntiva vs Disyuntiva (Múltiple cláusulas).
- Libre (premisa cláusula, cuando no se conoce su valor).
- Específicas (premisa cláusulas, es decir, específicamente cierta o falsa).

3.4.1 Propiedades de las reglas.

En las reglas también hay ciertas propiedades, éstas se pueden resumir en las siguientes [9]:

Nombre.- Es un nombre descriptivo y puede ser tan simple como: Regla1, Regla2, ... o bien un nombre que describa el propósito de la regla.

Premisa.- Cada regla consiste de una o más premisas, que en su conjunto, forman la Premisa de la regla, ésta consiste en cláusulas conjuntivas o disyuntivas. Cuando hacemos referencia al estatus de la regla, nos referimos a la colección de premisas individuales.

Conclusión.- Cada regla consiste de una o más conclusiones conjuntivas. Hay dos tipos: intermedias y finales; una conclusión intermedia, es aquella que sirve como premisa para o en otra regla; una conclusión final, es aquella que no aparece como premisa en alguna otra regla.

Notas y Referencias.- Es esencial documentar las reglas.

Factores de confidencia.- Cuando se emplea cierta incertidumbre en la decisión de alguna regla, podemos asociar factores de confidencia a cada regla.

Prioridad y costo.- Tales propiedades son empleadas como un medio para decidir durante el proceso de inferencia la regla específica a ser tratada. Típicamente, el proceso selecciona la regla con mayor prioridad y menor costo.

Preferencias de encadenamiento.- El proceso de inferencia envuelve cierto proceso de búsqueda, en algunos casos se mueve hacia adelante, es decir, de premisas (hechos) a conclusiones. En otros casos se mueve hacia atrás, es decir, de las conclusiones (hipótesis) a las premisas necesarias que se necesitan para llegar a la conclusión dada.

Estatus de la regla.- Durante la consulta, el estatus de cada cláusula y regla, esta sujeto a cambio. Llevar un registro de tales cambios es una parte esencial del proceso de inferencia, y debemos contar con la terminología empleada:

- La premisa de una regla es verdadera, cuando se le ha hecho una prueba y se determinó que la premisa se satisface.
- La premisa de una regla es falsa, cuando se le ha hecho una prueba y se determinó que la premisa no se satisface.
- Si la premisa de una regla es verdadera, se dice que dicha regla se activa o se dispara.
- Si la premisa de una regla es falsa, se dice que dicha regla es descartada o esta inactiva.
- Si una regla es disparada, entonces esto implica que la acción implicada por la conclusión es tomada, y los valores asociados con los atributos de la conclusión son asignados a cada uno.
- Una regla que ha sido disparada no esta activa por mucho tiempo. Esto es descartada o en algunos casos inactivada.
- Si una regla va a ser disparada, primero debe haber sido activada (triggered).
- Si una regla no ha sido descartada o disparada, esta regla es designada a ser como activa.

3.4.2 Agrupación de reglas.

Las bases de conocimiento llevan mucho tiempo. Como resultado se pueden convertir en una colección de reglas no relacionadas y quienes desarrollan tales bases desorganizadas, tendrán dificultades parta entender, mantener y validarlas. Por lo tanto, es necesario llevar cierta organización y usualmente, agrupar las reglas de acuerdo con sus conclusiones o atributos similares. Esto puede hacer más fácil la apreciación de varios factores que llevan a la misma conclusión y también detectar errores en la formulación de reglas o para hacer adiciones o borrarlas de la base de conocimiento.
El orden de agrupar reglas es, una vez que las reglas se han agrupado, cada grupo a su vez debe ser ordenado, típicamente este ordenamiento es determinado por los atributos de la conclusión, esto es, el primer grupo es de aquellas reglas que tienen metas finales en sus conclusiones. El siguiente nivel es asignado al grupo que sirva para concluir cualquier premisa del grupo más alto (el anterior) y así sucesivamente.

3.4.3 Fase de resolución de conflictos.

La fase de resolución de conflictos sirve para decidir cuál de las sugerencias se va a aceptar [9] [2]. A esta fase se le puede considerar como la estrategia de control y algunas de estas estrategias que se han utilizado son las siguientes:

- *No-duplicación.* No aplique dos veces la misma regla a los mismos argumentos.

- *Novedad.* Prefiera aquellas reglas que se refieren a elementos de la memoria de trabajo de reciente creación.

- *Especificidad.* De preferencia aquellas reglas que sean mas especificas. Por ejemplo, para las siguientes dos reglas, la segunda es la que se preferiría:
  Mamífero(x)⇒ añadir Piernas(x, 4)
  Mamífero(x) ∧ Humano(x) ⇒ añadir Piernas(x, 2)

- *Prioridad de operación.* Prefiera aquellas acciones que tienen mayor prioridad, según lo especificado por cierto sistema de calificación, por ejemplo, la segunda de las reglas siguientes posiblemente sea la que tenga más alta prioridad:

  TableroDeControl(p)∧Polvoso(p)⇒Acción(Desempolvar(p))
  TableroDeControl(p)∧LamparadeFusionEncendida(p)⇒Acción(Evacuar)

3.4.4 Sistemas de explicaciones.

Además de lograr simplemente una conclusión cuando enfrenta un problema complicado, un experto es también capaz de explicar, hasta cierto punto, el razonamiento que conduce a dicha conclusión. Un sistema experto debe diseñarse para brindar una facultad semejante. Esta es una potencialidad que generalmente esta ausente en los sistemas tradicionales de computación.

Típicamente la explicación consiste en una identificación de los pasos en el proceso de razonamiento y de una justificación de cada uno de ellos. El proporcionar esta potencialidad para comunicar esta información constituye esencialmente un subconjunto del problema del procesamiento del lenguaje natural [2][9]. El sistema debe acceder a un registro de los conocimientos que se emplearon en procesamiento, basándose en el esquema de representación de la base de conocimiento y traducirlo a una forma que sea aceptable por el usuario.
Para proporcionar los niveles críticos de explicación, el sistema debe identificar el nivel de conocimientos del usuario y entender cómo adaptar la explicación para acoplarla apropiadamente. Las facilidades de explicación en varios sistemas actuales, se limitan a simplemente listar las reglas que se utilizaron durante la ejecución.

3.5 Diseño de un Sistema Experto basado en Reglas.

En esta sección consideramos como desarrollo un sistema experto basado en reglas [3][8][9]. Resaltaremos los pasos principales seguidos que son típicos cuando se usa esta metodología, es importante abordar este tema, ya que el sistema de desarrollo que se implementará, utilizara esta metodología para la construcción de los sistemas expertos que serán montados en la Web.

Como en todos los sistemas, la primera tarea es entender el problema y obtener una idea general del mismo. Esta tarea envuelve la definición del proyecto, objetivo de los principales problemas y la forma en que el experto trabaja con la información disponible para derivar las recomendaciones.

Un sistema de Encadenamiento hacia Adelante, comienza con los datos del problema y dispara las reglas necesarias para inferir nueva información.

En un simple sistema de Encadenamiento hacia Adelante, la máquina de inferencia dispara las reglas cuyas premisas corresponden a la información de la memoria de trabajo. Hay ocho tareas principales que se deben hacer cuando se desarrolla un sistema de Encadenamiento hacia Adelante.

1. Definir el problema.
2. Definir los datos de entrada.
3. Definir la estructura del control de datos.
4. Escribir el código inicial.
5. Probar el sistema.
6. Diseñar la interfaz.
7. Expandir el sistema.
8. Evaluar el sistema.

A continuación se describe cómo hacer cada una de estas tareas para una aplicación específica de diagnostico automotriz.

Definición del problema.- El paso inicial es aprender acerca de diagnostico automotriz. Una opción sería conseguir un mecánico experimentado y considerarlo como un experto, otra opción es usar un manual de fallas, este tipo de manuales, contienen conocimiento de fallas de un experto, lo cual representa una buena fuente de obtención de conocimiento y a veces es mejor que tratar con un mecánico experimentado, es decir, podemos evitar los problemas que se presentan en la adquisición del conocimiento a través de entrevistas con el experto. Aunque en esta área de diagnostico automotriz, puede haber muchos problemas, éstos deben ser tratados individualmente según las tareas principales, así debemos especificar bien el problema en particular que será tratado.

Definir datos de entrada.- Todo sistema de encadenamiento hacia Adelante necesita obtener datos iniciales para comenzar. Así necesitamos definir reglas cuya única tarea sea preguntar sobre información acerca del problema. Estas son llamadas reglas de arranque, las cuales se disparan automáticamente cuando el sistema inicia. Estas reglas siempre están asociadas con la naturaleza del problema, estas reglas podrían tener la siguiente estructura:

SI Inicia el sistema
ENT. Pregunta el problema del auto.

Para dispararla, primero debemos asegurar que la que la tarea sea el Inicio es decir que esta se encuentre en la memoria de trabajo. La función "Pregunta", causa una pregunta que previamente asociamos con la expresión "problema del auto":

El motor no arranca.
El motor se jalonea a alta velocidad.

El motor trabaja erráticamente.

Después de que el usuario selecciona un problema en particular, el sistema direcciona la solución del problema a un área apropiada.

Definir la estructura del control de datos.- En teoría, un sistema de Encadenamiento hacia Adelante trabaja disparando reglas cuyas premisas están contenidas en la memoria de trabajo; usando estas reglas que se disparan, el sistema infiere nueva información acerca del problema o bien realiza otra tarea específica. En aplicaciones pequeñas, este tipo de control de flujo de disparo de reglas, puede proveer resultados adecuados. Como sea en muchos sistemas de Encadenamiento hacia Adelante necesitarás incluir con cada regla, una premisa que ayude al control cuando una regla puede ser disparada, por ejemplo:

IF Tarea es .....
AND A
ENT. Inferir o hacer algo.

Esta solo se dispara si "Tarea es .....", y A son verdaderas y así ayuda a mantener un control sobre el proceso natural de inferencia en encadenamiento hacia Adelante.

Escribir el código inicial.- El propósito de la codificación inicial en un sistema de Encadenamiento hacia Adelante, es determinar, si efectivamente capturamos el conocimiento del problema en una buena estructura de reglas. Una buena estructura de reglas no solo provee resultados correctos, sino también una plantilla para seguir con el desarrollo de otras reglas.

Prueba del sistema.- La siguiente tarea es probar el pequeño conjunto de reglas; podemos asumir que "Tarea es....." ha sido inicializada en la memoria de trabajo, esto causa que el comienzo que dispara la Regla1 y el sistema pregunta "cuál es el problema con el auto", aquí el sistema busca en todas las reglas que se relacionan con "problema del auto" y encuentra a Regla2 y Regla3 que proporcionan elementos adicionales al menú de preguntas. Después podemos expandir el sistema para considerar problemas con la gasolina o el sistema de frenos, simplemente agregando las reglas apropiadas.
Así el sistema ahora hace la pregunta

Sistema: Cual es el problema con el motor
a.- No arranca.
b.- Trabaja erráticamente.

Usuario: No arranca

Si el usuario responde la opción "a", esta respuesta dispara a Regla2 y coloca a la tarea "Probar el sistema de arranque" en la memoria de trabajo, esto causa que se dispare Regla4 con la siguiente pregunta:

Sistema: De marcha al motor, y observe el movimiento de este.
a.- No gira, o gira lentamente.
b.- Gira de manera normal.
Usuario: No gira, o gira lentamente.

Con esto se dispara Regla5 que da como resultado colocar "problema con el sistema de arranque" en la memoria de trabajo y "Tarea es probar las conexiones de la batería" que dispara a Regla7 con la pregunta:

Sistema: Coloque un desarmador entre el poste de la batería y la abrazadera, conecte las luces y vea la intensidad de éstas. ¿Qué pasa con las luces? :
a.- Aumenta la intensidad.
b.- No encienden
c.- No aumenta la intensidad.
Usuario Aumenta la intensidad.

Esto dispara a Regla8, resultando en la conclusión "El problema es en las conexiones de la batería, siguiendo esto, ya no hay mas reglas que disparar y el sistema se detiene.

Diseño de la interfaz.- ya que tenemos el conjunto de reglas trabajando, el siguiente paso es ¿Cómo hacer la interfaz del sistema?. Es recomendable construir la interfaz después de que la base de conocimiento haya sido completada. Esta interfaz es un componente muy importante del sistema, y puede diseñarse paralelamente al desarrollo de la base de conocimiento. Y no como una tarea posterior ya que la forma en que se diseña la estructura de la base de conocimiento, es influenciada por la forma en que diseñas a la interfaz.
En nuestro ejemplo, se ve claramente que la interfaz esta centrada en la función de preguntas que se hacen al usuario.

Interfaz gráfica dinámica. Muchos de los primeros sistemas expertos fueron diseñados para interactuar con el usuario en modo texto únicamente. Muchos shells de ahora proveen elementos gráficos para construir a la interfaz. La base de conocimiento contiene la información que será desplegada en la interfaz y las reglas de encadenamiento hacia Adelante que trabajan con dicha información.

Formato de la interfaz. La clave para el diseño de una buena interface, es la consistencia. Para cada pantalla material similar deberá ser colocado en las mismas posiciones, esto permite al usuario desarrollar un modelo mental de donde esta la información.

Contenido de la pantalla. Hay dos tipos principales de contenidos: pantallas de despliegue, y pantallas de preguntas.

Pantallas de despliegue.- Su propósito es presentar información al usuario y hay tres tipos:

- *Introducción:* Debes incluir oraciones que explique el objetivo del sistema. Una pequeña discusión del problema dado, también puedes discutir la aproximación que el sistema experto tomará para resolver el problema y así prepararlo para lo que sigue. Hay dos funciones típicas que debe haber en esta pantalla, una para empezar la sesión y otra para salirse.
- *Resultados inmediatos:* Para desplegar al usuario acontecimientos importantes alcanzados por el sistema. Hay tres funciones de control básicas que debes proveer en esta pantalla:
  Reiniciar: La sesión en el evento donde quieras cambiar ciertos valores.
  Salir: Para salir del sistema.
  Continuar: Simplemente resume la sección.

- *Conclusión:* Presenta al usuario las recomendaciones finales, la forma de hacerlo depende de la aplicación, puede ser una simple recomendación, una lista de recomendaciones, o una forma con muchas piezas de información que, dio el usuario y que, el sistema infirió, además las dos funciones típicas que se agregan en esta pantalla son: *Salir* y *Reiniciar.*

Pantalla de preguntas.- Es usada para obtener del usuario información acerca del problema, tiene tres partes básicas:

- *Porción del texto que contiene la pregunta:* Esta debe escribirse al nivel del usuario.
- *Respuesta:* Hay dos formas básicas en que el usuario puede responder. Menú de selección que es la más cómoda, y texto directo que es donde puede haber más errores.

Expandir el sistema. Siguiendo la prueba exitosa de nuestro conjunto de reglas, la siguiente tarea es expandir el conocimiento del sistema. Esto es desarrollar reglas adicionales, para nuestro ejemplo de diagnóstico automotriz, sería hacer las reglas para otro tipo de problemas por ejemplo relacionadas con la suspensión, el sistema de enfriamiento, frenos etc.

La expansión también debe incluir el desarrollo de pantallas de la interfaz y reglas que puedan desplegar varias pantallas; así nuestro sistema en desarrollo ya será un prototipo y estará listo para la evaluación.

Evaluar el sistema. Esta tarea esta relacionada con la prueba del prototipo mediante alguna prueba o pruebas reales, normalmente se lleva a cabo con ayuda del experto humano o siguiendo el manual. Para verificar que nuestro sistema este trabajando apropiadamente, podemos dar respuestas apropiadas a cada punto de decisión y ver si el sistema llega a la misma falla que escogimos.

Instrucciones futuras. Ya que hemos construido y probado exitosamente nuestro sistema, el siguiente reto es continuar con su expansión. Para nuestro problema, hacemos esta tarea simplemente continuando con el desarrollo de pasos adicionales encontrados en nuestro manual.

Resumen.

- El diseño de un sistema experto con Encadenamiento hacia Adelante, es un proceso altamente iterativo
- Debe ser probado inmediatamente después de la introducción de nuevas reglas.
- El proceso de Inferencia de encadenamiento hacia adelante es preferible sobre la de encadenamiento hacia atrás si las metas no son conocidas a priori, o si son muchas.
- Diseñar un sistema experto con encadenamiento hacia atrás empieza con la definición de las metas del sistema, mientras que en el proceso de diseño de un sistema experto con Encadenamiento hacia Adelante empieza con la definición de los datos iniciales.
- Las reglas usadas en un sistema con encadenamiento hacia adelante siempre contienen premisas que pueden dirigir la solución del problema con el fin de mantener un mejor control sobre el proceso de inferencia.
- El diseño de la interfaz del sistema debe empezar temprano en el proyecto, y desarrollarla en paralelo con el desarrollo de la base de conocimiento.
- Una interfaz dinámica contiene objetos gráficos que permiten al usuario observar el control y la operación del sistema experto.
- La clave para desarrollar una buena interfaz de usuario, es la consistencia

Capítulo IV

Implementación del sistema.

Introducción.

En el presente capítulo se describe como se implementó el sistema propuesto, se establecen y describen los elementos utilizados, así como la manera en que se interrelacionan para lograr los objetivos planteados.

Se define la Arquitectura implementada, así como la operación de cada uno de los elementos que la conforman.

Se describe la implementación del ambiente de desarrollo para los sistemas expertos, esto es el programa que incluye cada una de las fases de implementación para dichos programas.

Se describen cada uno de los componentes del ambiente de desarrollo y su funcionamiento para lograr cada una de las fases de desarrollo de sistemas expertos basados en reglas.

Al final, se describen las herramientas de programación y lenguajes de alto nivel que se usaron para este propósito y el porqué fueron seleccionados.

4.1 Descripción general.

Como ya se mencionó, el objetivo principal de este trabajo es implementar una arquitectura para el desarrollo, implementación y distribución de sistemas expertos a través de la World Wide Web; con base a los requisitos de este sistema que se detallaron en el capítulo II, hemos establecido los elementos necesarios para la implementación de nuestra arquitectura, en primer lugar, tenemos los elementos que están constituidos por el hardware, como elementos físicos (Dispositivos eléctricos y electrónicos), específicamente computadoras y sistemas de comunicación que conforman la parte del sistema correspondiente a la red; en segundo lugar, se encuentran los elementos de software (Programas de aplicación, programas de desarrollo y sistemas operativos) que constituyen la parte operacional del sistema y que para poder trabajar necesita de la primera.

En este capítulo también se describe la función de cada elemento, del ambiente desarrollo y de los programas de desarrollo del mismo (es decir lenguajes de programación), además de la justificación del porqué y cómo fue seleccionado cada uno de estos.

Para la implementación de nuestra arquitectura propuesta, se contemplan los siguientes elementos:

1. Una pequeña red de computadoras, la cual incluye un servidor conectado a Internet para así tener servicio a la World Wide Web.

2. Un sistema operativo de red que permite levantar y ofrecer servicios de Internet; para este caso se seleccionó Windows Advanced Server 2000 como sistema operativo de red.

3. Tres computadoras conectadas al servidor, con sistemas operativos en ambiente Windows de versión diferente, estas máquinas tienen diferentes funciones como: Desarrollo, pruebas de depuración y pruebas de usuarios finales.

4. Ambiente de desarrollo, una aplicación con las diferentes etapas de desarrollo de sistemas expertos basados en reglas, éste fue implementado mediante un lenguaje de programación orientado a objetos. Se seleccionó el lenguaje Java por su potencialidad para manejo de datos y programación de aplicaciones que pueden correr en Web.

5. Herramientas de programación y evaluación, que comprenden: Lenguajes de programación de alto nivel como pueden ser: Visual Basic, Java, C++ entre otros.

6. Programas de desarrollo y evaluación de sistemas expertos, los cuales se hacen mediante la aplicación del punto 4 y son formateados para obtener programas de sistemas expertos listos para ser ejecutados en la World Wide Web.

4.2 Diseño e Implementación.

A continuación se detallan cada uno de los elementos que conforman nuestra arquitectura propuesta, empezaremos con la parte física, es decir, elementos de Hardware, después seguiremos con los elementos de software que incluyen los programas de aplicación finales.

La figura 4.1, nos muestra nuevamente la estructura del ambiente de desarrollo para la arquitectura propuesta.

Como se puede observar, hay una interrelación entre: el ambiente de desarrollo, la base de conocimiento, la máquina de inferencia y la parte de pruebas y verificación, todas estas funciones se controlan y llevan a gracias a la interfaz de usuario que además, nos sirve también para preparar los sistemas expertos para su operación en la Web.

Este estructura, nos permitirá llevar a cabo las diferentes etapas para el desarrollo de sistemas expertos basados en reglas y posteriormente, permitir que los usuarios finales hagan consultas a éstos a través de la Web.

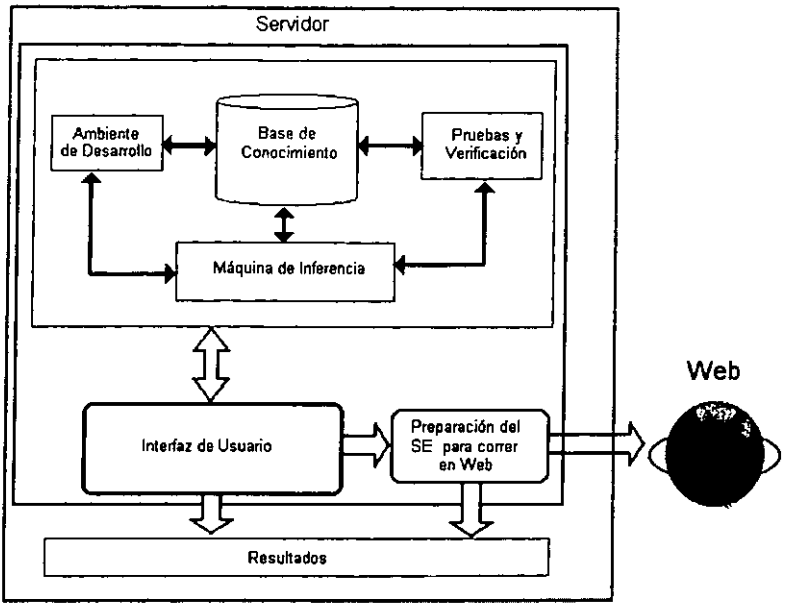


Figura 4.1 esquema del ambiente de desarrollo para la arquitectura planteada.

4.2.1 Red de computadoras

Actualmente las redes de computadoras se encuentran en casi todos lados, desde pequeñas empresas, casas particulares hasta grandes corporativos y en Universidades e Institutos de Investigación; además cada día más computadoras se enlazan a la red internacional mejor conocida como Internet [10].

Para nuestra arquitectura, necesitamos de tres computadoras y un servidor, con lo cual tenemos una minired; el servidor está conectado a Internet, a través de un enlace con el servidor de la UNAM (Universidad Nacional Autónoma de México). La estructura física de nuestra minired y que sirve para nuestros fines se muestra en la figura 4.2.

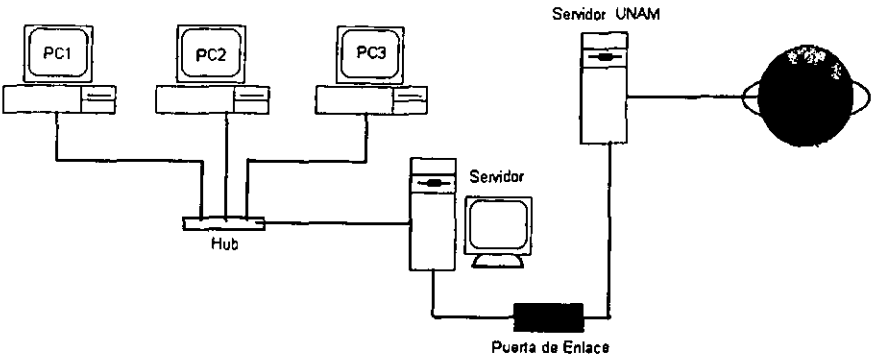


Figura 4.2 Estructura de la minired para desarrollo de sistemas expertos.

Como se puede observar, tenemos tres computadoras personales, conectadas al servidor mediante un concentrador; el servidor, se conecta directamente a la red de la UNAM; el dominio que le fue asignado por el DNS de la UNAM es: *wizard.cinstrum.unam.mx* y que además, este servidor y la minired que tenemos, forman parte de la red del Centro de Instrumentos de la UNAM.

En el proceso de conexión de esta red, se realizaron las siguientes tareas.

Configuración y preparación del servidor, para la carga del sistema operativo de Red. Para esta tarea, se utilizo una computadora tipo PC con las siguientes características

Procesador Pentium III a 500 Mhz.
128 Mbytes de Memoria RAM.
Disco duro de 20 Gbytes.
Unidad de lectura para CD de 52X.
Unidad de discos de 3.5
2 tarjetas de comunicación Ethernet 100/10 Mbps
Monitor a color Super VGA 15"
Fax Módem 56 Kbps

Nota (Este equipo y los que se describen a continuación, fueron proporcionados por el Centro de Instrumentos de la UNAM).

Sistema Operativo para Red. Una vez que se terminó la configuración del equipo, se procedió a la instalación del sistema operativo de Red (Windows 2000 Advanced Server), este sistema cuenta con los elementos necesarios para que la instalación se lleve a cabo de una manera rápida y eficiente.

Levantamiento de los servicios de red en el servidor. Los servicios de Red que se integraron en el servidor, son todos los necesarios para la integración de una pequeña minired:

DHCP.
Cliente para Redes Microsoft.
Cliente para redes Netware.
Protocolos NetBeui.
Protocolos TCP/IP.
Protocolos de Acceso telefónico a redes.
Internet Infomation Server.
Servidor WEB.
Servidor FTP.
Servidor Gopher
DNS Servidor de Nombres de Dominio (Local)

Asignación de nombre de dominio para Internet. El nombre de dominio asignado fue *wizard.cinstrum.unam.mx*, con el cual se identifica este servidor en la red de la UNAM y además esta registrado en Internet bajo la dirección IP: 132.248.36.72, esta dirección IP, fue asignada a la tarjeta de conexión externa a la red de la UNAM. A manera de explicar esta conexión, tenemos el esquema de la figura 4.3.

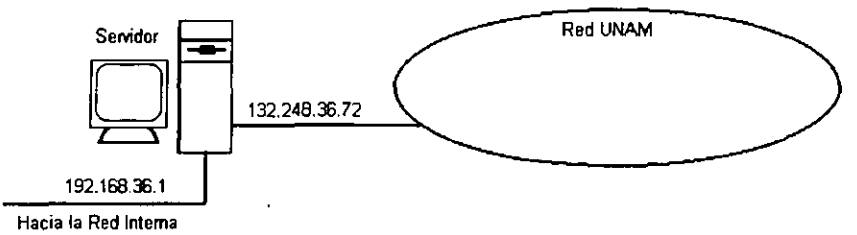


Figura 4.3 Direcciones IP asignadas al servidor wizard.

Como se puede observar en la figura 4.3, nuestro servidor tiene dos puertos de conexión a Ethernet, uno de ellos le permite enlazarse a la red de la UNAM, y el otro es para los servicios internos de la red, uno de éstos, es la compartición de acceso a Internet para las demás computadoras, éste se hace a través de esta conexión.

Por otro lado, también se tiene una conexión hacia las computadoras cliente de este servidor y las cuales conforman a la minired; para esta minired se estableció una configuración de red tipo C, y se asignó un intervalo de direcciones IP que comprende las direcciones 192.168.36.0 a 192.168.36.255; la asignación de estas direcciones, se lleva a cabo mediante el servicio DHCP en el servidor wizard.

4.2.1.1 Configuración del concentrador a utilizar.

El concentrador utilizado, es un Hub a 10 Mbps de 8 puertos de conexión, se utilizó un modelo de la marca 3Com que no necesita configuraciones muy complejas, simplemente se conecta a la energía eléctrica, y se procede a hacer las conexiones de comunicación Ethernet a las computadoras cliente, estas conexiones se realizaron con cable UTP nivel 5, la forma en que quedó organizada la configuración de las conexiones se muestra en la figura 4.4.

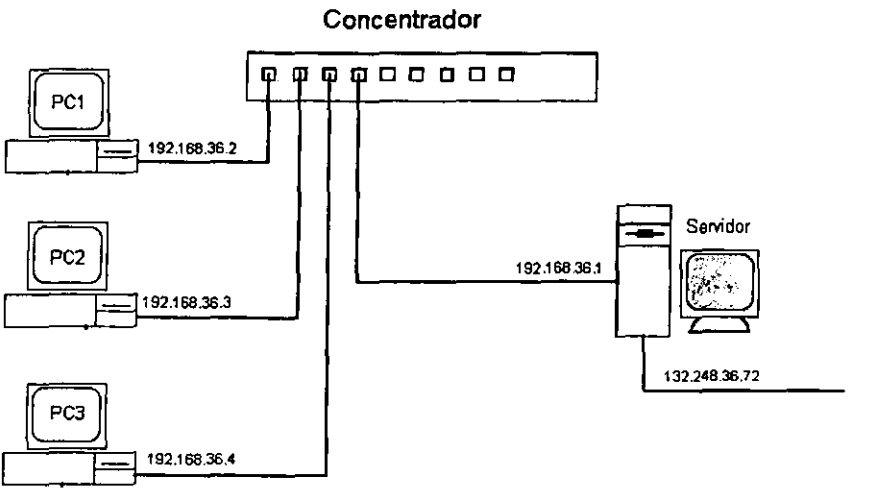


Figura 4.4 Conexiones Ethernet y direcciones IP para la minired.

4.2.1.2 Configuración y preparación de las computadoras.

Para complementar el resto de equipos de la red interna, necesitamos tres computadoras más, que serán las estaciones de trabajo de la minired, estas computadoras tienen las siguientes especificaciones:

Procesador Pentium, PentiumII o PentiumIII de 300 a 500 Mhz.
64 Mbytes de Memoria RAM y 128Mbytes para la PC de desarrollo.
Disco duro de 10 Gbytes.
Unidad de lectura para CD de 52X.
Unidad de discos de 3.5
1 tarjetas de comunicación Ethernet 100/10 Mbps
Monitor a color Super VGA 15"

A cada computadora, se le instaló un sistema operativo Windows de diferente versión, esto es con el fin observar la operación de nuestros sistemas en

diferentes versiones de estos sistemas operativos; la asignación de cada uno quedó de la siguiente forma:

Computadora PC1	Windows 95
Computadora PC2	Windows 98
Computadora PC3	Windows 2000

Las tareas asignadas a cada una de las computadoras quedan de la siguiente manera.

Computadora	Tareas
PC1	Usuario
PC2	Desarrollo de sistemas expertos, pruebas y depuración.
PC3	Diseño e implementación de las aplicaciones.

Así, de esta manera en el equipo denominado PC1 se harían consultas directamente sobre los sistemas expertos ya terminados, mediante enlaces directos a través del acceso por medio de Internet al servicio Web del servidor, esto permite hacer las pruebas finales de estos sistemas y así determinar el comportamiento y operación real de dichos sistemas, además de que nos permite también determinar si dichos sistemas necesitan modificaciones en caso de que se detecten anomalías en el proceso de ejecución en Web.

El equipo denominado PC2, es el que tiene instalada la aplicación de desarrollo de sistemas expertos, aquí es donde se construyen siguiendo las diferentes etapas de desarrollo, una vez realizadas las pruebas y depuración, se montan en el servidor Web para que puedan ser consultados por los usuarios.

El equipo denominado PC3 se utilizó principalmente para la programación de las aplicaciones de desarrollo de sistemas expertos (aplicaciones que se instalaron en PC2) para ello, se instalaron lenguajes de programación como Java, Visual Basic, Visual Café y shells comerciales para el desarrollo de sistemas expertos, entre otros.

Los shells comerciales como Exys, CruXpert, CiaAgent, fueron de mucha utilidad, para el desarrollo de nuestras aplicaciones, ya que proporcionaron mucha información y ayuda para la implementación del sistema.

Después de trabajar y evaluar cada uno de los lenguajes de programación, se decidió usar lenguaje Java para el desarrollo de las aplicaciones, utilizamos JBuilder2 que es una herramienta de desarrollo visual, para de sistemas en lenguaje Java.

4.2.2 Implementación del ambiente de desarrollo.

En esta sección, se describen cada una de las partes del programa para el ambiente de desarrollo de sistemas expertos, estos subsistemas, son programas

de computadora que se desarrollaron en lenguaje Java, la mayoría de estos, son subprogramas a los cuales se les llama clases; primero se describen las clases que sirven para implementar cada una de los componentes que sirven para el desarrollo de sistemas expertos basados en reglas (como: Reglas, Variables, procesos de Inferencia etc.) después, se describen los elementos que conforman la interfaz gráfica de usuario y cómo se relacionan con los primeros para el desarrollo de sistemas expertos basados en reglas.

En esta sección, no se presenta mucho contenido del código fuente, para complementar la explicación de cada una de las clases, se recomienda observar el código completo de cada clase en el apéndice A.

Dado que estamos trabajando con un lenguaje orientado a objetos todas las variables, e incluso las mismas clases son consideradas como objetos [13], a lo largo de esta sección, continuamente se estará hablando en términos de objetos como elementos de programas en Java, los cuales pueden ser tanto variables como las mismas clases

4.2.2.1 Reglas.

Dado que nuestros sistemas expertos que se implementarán, se basan en la representación de conocimiento mediante reglas, se tiene una parte que sirve para definir cada una de las reglas que formarán la base de conocimiento del sistema experto en desarrollo, esta parte se realiza mediante una clase (así como todas las demás) en código Java; la clase pública **Rule** (en el apéndice A de este documento se presenta el código fuente de ésta y las demás clases que se describen en esta sección), tiene los siguientes atributos.

- *Nombre de la base de reglas.*- Este parámetro, es un objeto identificador que le permite a la regla hacer referencia a la base de reglas a la cual pertenece.

- *Nombre.*- que sirve para identificar a la regla, este nombre se le asigna al momento de la creación, y puede ser cualquier cadena de caracteres que servirá para identificar a la regla; pueden ser asignados nombres tan simples como Regla1, Regla2 etc. o bien nombres altamente descriptivos como "Verificación del estado de la máquina uno", "comparación del nivel en el medidor de flujo" etc.
- *Premisas* (Arreglo de Cláusulas). - Este es un arreglo del tipo vector de objetos, que almacenan el conjunto de premisas que pertenecen a la regla (más adelante se explican la estructura de las cláusulas) de antemano se asume que este conjunto de premisas, están ligadas por un conector lógico del tipo "Y" (conjunción), así cada una de las cláusulas constituye una premisa de la regla en cuestión, además una misma cláusula o premisa puede pertenecer a varias reglas, de aquí la ventaja de que se haya estructurado en un arreglo de objetos del tipo cláusula, ya que al ser objetos del mismo tipo, estos se convierten en referencia de un mismo objeto, lo cual representa un ahorro de

memoria al evitar tener dos o más objetos iguales. El número máximo de premisas que se pueden tener por cada regla es de 10, consideramos que es suficiente para los fines de nuestro trabajo, aunque con la simple modificación del código fuente de esta clase, se puede aumentar el número de premisas permitidas.

- *Consecuente.*- Este también es un objeto del tipo cláusula, y es el consecuente o la conclusión que se obtiene cuando todas las premisas de la regla se cumplen, es decir cuando estas son verdaderas.
- *Truth (Valor de tipo booleano).* - Este parámetro es de tipo booleano, e indica cuándo una regla puede o no ser evaluada, este sirve o nos permite usar un valor nulo para indicar que el valor Truth de la regla no puede ser determinado, porque alguna de las variables a las que hace referencia o alguna de las cláusulas o premisas, también es nula o indefinida.
- Fired *(Valor de tipo booleano).*- Este parámetro también es de tipo booleano, y sirve para indicar cuándo una regla ha sido disparada ó no.

El constructor básico para la definición de una regla, es el siguiente:

```
Rule(RuleBase, Nombre, Clause1, Clause2,.....,
ClauseR)
```

El constructor de la clase Rule, recibe como parámetros: la Base de Reglas, hasta diez cláusulas antecedentes o premisas y un consecuente, también de tipo cláusula.

Además de sus atributos, la clase pública Rule cuenta con varios métodos que definen el comportamiento de una regla construida con esta clase. Entre éstos, tenemos métodos para el control de los procesos de inferencia y métodos de despliegue de datos. A continuación se da una breve descripción de uno.

- *numAntecedents()*: nos da el número de premisas que tiene la regla (máximo 10).
- *checkRules():* prueba todas las reglas que hacen referencia a una determinada variable.
- *fire():* dispara una regla cuando todas sus premisas son verdaderas.
- *backChain():* determina si una regla puede ser verdadera ó falsa, por medio de recursividad, al probar si cada una de sus premisas son verdaderas. Este método funciona en conjunto con el método de encadenamiento hacia atrás de la clase **RuleBase.**
- *display():* despliega las reglas en forma de texto legible, en campos específicos de la interfaz de usuario.
- *waitForAnswer():* despliega un cuadro de diálogo, para preguntar por variables a las cuales no se conoce su valor actual.

4.2.2.2 Cláusulas

Cada regla en la base de conocimiento, está formada por cláusulas, las cuales a su vez, están formadas por una variable y un valor determinado. Las cláusulas también son objetos usados en las premisas y consecuentes de una regla; cada cláusula tiene tres parámetros principales que se organizan de la siguiente manera para construir este objeto:

Variable-Condición-Valor

Como se puede observar, cumple con las características de la tripleta Objeto-Atributo-Valor (OAV descritas en el capítulo II) [9], donde la Variable corresponde al objeto, las condiciones a alguna de las asignaciones de igualdad, diferencia, mayor o menor que, u operadores de comparación de patrones en lenguaje natural, como "es", "es un", "tiene", "tiene un", "es mayor" etc.

Cada cláusula puede ser premisa en una o más reglas, y también pueden ser consecuentes en una o más reglas, las cláusulas se definen mediante la clase pública en Java denominada **Clause** [4], esta clase tiene los siguientes atributos:

- *RuleVariable.*- Este es un objeto que hace referencia a una variable de regla que será comparada con el objeto valor del lado derecho.
- *Condición.*- Es un parámetro que permite hacer una comparación de: Igualdad, diferencia, mayor que, menor que o comparación de patrones.
- *Valor.*- Este puede ser una cadena de caracteres o un número, y representa el valor que deberá tener asignado la variable, para: que la cláusula sea considerada verdadera, cuando dicha cláusula es una premisa de una regla, ó el valor que deberá ser asignado a la variable cuando esta cláusula es el consecuente de la regla.
- *Consecuente valor booleano.*- Este es un parámetro booleano que indica si dicha cláusula aparece en el consecuentes o en el antecedente de la regla. Un valor verdadero para este parámetro, indica que esta cláusula es consecuente de una regla, mientras que un valor falso indica que es antecedente de la regla
- *Truth valor booleano.*- Nos indica si la cláusula es verdadera, falsa o desconocida, esto es si por ejemplo tenemos la cláusula: Variable = ValorX un valor Truth=Verdadero, nos indica que en la memoria de trabajo, Variable si tiene asignado a ValorX, por otro lado un valor Truth=Falso nos indica que el objeto Variable no tiene asignado a ValorX, y por último, un valor Truth=nulo nos indica que Variable no tiene asignado ningún valor es decir que ésta no se encuentra en la memoria de trabajo.

Como ejemplo, consideremos la siguiente regla:

SI: Número_llantas = 4
y: Motor = Si.
Ent.: Tipo_Vehiculo = Automóvil.

Observamos que la regla anterior está formada por tres cláusulas, las cuales involucran a tres variables Número_llantas, Motor y Tipo_Vehiculo; el proceso de inferencia que se lleva a cabo para validar esta regla trabaja de la siguiente manera: Primero se analiza el valor del atributo consecuente de la cláusula, que por la estructura de la regla sabemos que tiene asignado *false*, por lo tanto esta cláusula es una premisa (igual que la siguiente); en segundo lugar, el proceso busca en la memoria de trabajo el valor asignado para la primera variable (*Número_llantas*), sí éste tiene un valor nulo o diferente de 4, la regla es marcada como invaluable (valor Truth=nulo) y es descartada, por otro lado sí el valor de la primera variable en la memoria de trabajo corresponde al valor 4, esta cláusula es marcada como verdadera y su valor Truth también es verdadero, y se pasa entonces a analizar la siguiente premisa o antecedente, de la misma manera sí la variable *Motor* no está en la memoria de trabajo o si está pero tiene un valor diferente a "*Sí*", la regla será descartada, ahora bien sí nuevamente se cumple que la variable *Motor* está en la memoria de trabajo y tiene asignado el valor "*Sí*", esta cláusula es marcada como verdadera Truth=verdadero y se procede con la siguiente cláusula, que por la estructura de la regla sabemos que tiene un valor en consecuente igual a verdadero, por lo cual, a la variable *Tipo_Vehiculo* le es asignado el valor *Automóvil*, y este nuevo dato se agrega a la memoria de trabajo; además los valores *Truth* y *Fired* de la regla se les asigna también el valor verdadero indicando que la regla ha sido disparada. El constructor básico de este clase es:

```
Clause(RuleVariable, Condición, Valor)
```

Este constructor, toma una variable de regla para el lado izquierdo (objeto) un objeto condición (Atributo), y una cadena de caracteres para el lado derecho (valor). La cláusula se registra a sí misma mediante el objeto variable de regla, que es también una referencia a un objeto global, así cuando este objeto global cambia de valor, no se necesita reasignarlo a la cláusula, ya que por la misma característica y manejo de memoria con Java, esto se hace automáticamente. Cuando se declaran inicialmente las cláusulas, a estas se les asigna un valor false a sus consecuentes, lo cual indica que al inicio todas pueden ser premisas y/o antecedentes de las reglas.

La clase pública Clause, también tiene métodos para el manejo de datos:

- *addRef():* agrega las reglas a las cuales pertenece dicha cláusula.
- *check():* este método se usa para determinar si la cláusula es verdadera ó falsa.
- *isConsequent():* este método determina si una cláusula es ó no consecuente o premisa de una regla.

4.2.2.3 Condiciones

Esta es una clase auxiliar de la clase **Clause**, lo que hace es tomar un objeto de tipo cadena de caracteres que es una representación de una prueba condicional, esta cadena la convierte a un código para ser usado en la comparación de variables, las cadenas que se declararon para esta función son:

Igual, es un, es :	=
Mayorque:	>
Menorque:	<
Diferente a:	!=

Los únicos métodos que se usan en esta clase, son para fines de despliegue en la interfaz de usuario, estos métodos son:

- *asString():* regresa las reglas a las cuales pertenece dicha cláusula.
- *check()*: este método es el que verifica sí la cláusula es verdadera ó falsa.
- is*Consequent():* este método determina sí una cláusula es ó no consecuente de una regla.

4.2.2.4 Variables

Esta es una clase general que define las características de un objeto que se comporta como una variable, tanto en el ámbito de programación como en su uso en los sistemas expertos basados en reglas, es decir que este objeto puede tomar diferentes valores permitidos durante el proceso de inferencia.

La clase pública **Variable**, tiene una serie de parámetros que sirven para determinar su comportamiento durante el proceso de inferencia, estos parámetros son:

- *Nombre.-* Para identificar a dicha variable.
- *Valor.-* Que es una objeto de tipo cadena, y nos indica el valor (de uno de los valores permitidos) que en ese momento, tiene asignado dicha variable.
- *Labels.-* Que es un arreglo de cadenas de caracteres, donde estos representan todos los valores permitidos que puede tomar dicha variable.

El constructor de Java, permite declarar variables sin parámetros y con parámetros, estos son:

`Variable () { }` Sin parámetros

`Variable(String nombre) {.....}` Con parámetros inicializa cada uno.

La clase pública Variable, tiene dos métodos para el control y manejo de datos, éstos se describen brevemente a continuación.

- *setLabels():* este método toma una cadena de texto con todos los valores separados por un carácter de control "&" y cada elemento es colocado dentro de un vector de valores permitidos para dicha variable; estos valores, también son colocados en varios campos de la interfaz de usuario.
- *getLabels():* hace el proceso inverso al anterior, este método regresa una cadena de texto con los valores de una variable, separados por un carácter el control "&".

4.2.2.5 Variables de Regla.

Para llevar a cabo el procesamiento de las reglas, se tiene la subclase **RuleVariables,** que hereda casi todo de la clase **Variable** pero, se le agregó un comportamiento específico, para que estos objetos sean, considerados como variables de reglas [4]. Esta clase provee el soporte necesario para las variables usadas durante el proceso de inferencia.

El constructor de esta clase, toma el nombre de la variable como único parámetro y hereda el comportamiento simbólico y discreto de la clase base **Variable**.

Un nuevo parámetro de esta clase es **ClauseRefs** (de tipo Vector) que guarda las referencias a todas las cláusulas que relacionan a dicha variable de regla. Las instancias de cláusula, se registran por si mismas llamando al método **addClauseRef().** Hay muchos métodos que se sobreponen a los de la clase base, pero también se agregaron nuevos métodos.

Un nuevo método **setValue()** no solo establece el valor de la variable, también llama al método **UpdateClause()** que interactua con cada cláusula que hace referencia a dicha variable de regla y reexamina su valor Truth mediante el método **check().**

La cadena que se almacena en la variable de clase **promptString,** almacena el texto que será desplegado cuando, durante el proceso de inferencia se le solicita al usuario información acerca de dicha variable de regla o cuando, la variable corresponde a una recomendación final obtenida durante el proceso de inferencia.

La variable de clase **ruleName**, guarda el nombre de la regla que tiene asignado este valor de la variable de regla, y cuando la regla se dispara, esta llama al método **setRuleName()** respectivamente, que guarda en una variable de clase **ruleName,** el nombre de la regla que se disparó.

El método **askUser()** llama a su vez al método **waitForAnswer()** en la clase principal que abre un cuadro de diálogo donde se pide al usuario que le asigne un valor para dicha variable; esto ocurre durante el proceso de inferencia, aunque solamente es usado cuando en el proceso de inferencia se aplica encadenamiento hacia atrás.

4.2.2.6 Base de Reglas.

Esta clase define un conjunto de variables de regla (**RuleVariable**) y Reglas, junto con los métodos de inferencia de alto nivel para encadenamiento hacia adelante y encadenamiento hacia atrás.

La clase RuleBase, tiene como parámetros principales: Un nombre **name** para identificar a la base de reglas, una lista de variables **variableList** que contiene todas las variables de regla referenciadas por las reglas y una lista de reglas **ruleList** que contiene todas las reglas.

La clase pública RuleBase, usa métodos que en su mayoría son para despliegue de información en la interfaz de usuario.

Dos métodos de gran interés y tal vez los más importantes, son los que realizan los procesos de inferencia. Estos métodos se llaman durante la operación del sistema experto; dado que son muy importantes, se explicarán a detalle la función de cada uno de ellos.

En los métodos de encadenamiento hacia atrás y encadenamiento hacia adelante, la clase **RuleBase** usa métodos y datos que son usados por estos algoritmos de inferencia y que son descritos a continuación.

4.2.2.7 Encadenamiento hacia Adelante.

La implementación de encadenamiento hacia adelante, usa métodos propios de la clase **RuleBase** y la clase **Rule**. El método de encadenamiento hacia adelante dentro de la clase **RuleBase**, contiene un control lógico principal. El método, primero asigna espacio de memoria para el vector de reglas que están en conflicto (reglas que pueden ser disparadas). Hay un método llamado **match()**, que es llamado con un parámetro booleano "true" para forzar una prueba inicial de todas las reglas que pertenecen a la base de reglas, este método regresa un nuevo vector inicial de reglas en conflicto, **conflictRuleSet**, que contiene solo las reglas que son activadas y que pueden ser disparadas. Una vez hecho esto, se lleva a cabo un ciclo donde se prueba cada una de las reglas del conjunto en conflicto, para ver si pueden ser disparadas, esto ocasiona que se genere nueva información que es almacenada en la memoria de trabajo; este ciclo corre hasta que ya no hay mas reglas dentro del conjunto en conflicto, es decir cuando se han evaluado todas y cada una de las reglas.

De manera general, este proceso trabaja de la siguiente manera: Dentro del ciclo, primero llamamos al método **selectRule** al cual le pasamos el vector (conjunto) de reglas en conflicto **conflictRuleSet** como único parámetro. Después, el método **selectRule** realiza una estrategia de solución de conflictos y regresa una sola regla a disparar, esto se realiza llamando al método **Rule.fire** que hace la asignación del consecuente correspondiente inmediatamente después de esta

asignación, se reexaminan todas las cláusulas y reglas que hacen referencia a la variable actualizada.

Aunque no es una implementación del algoritmo de Rete [15], este método limita la cantidad de cláusulas a probar. Con la lista de variables actualizada, llamamos al método **match()** nuevamente, pero esta vez le pasamos el valor booleano false lo cual le indica al método que únicamente compare o analice las reglas que no están en conflicto, esto con el fin de que con base a la nueva información que se generó al disparar una regla, puede ocasionar que nuevas reglas estén en la posibilidad de ser disparadas y sean agregadas al conjunto de reglas en conflicto. A continuación veremos como trabaja cada uno de estos métodos con más detalle.

Método **match()** se llama haciendo referencia a la clase *RuleBase*, que como ya dijimos, es a la que pertenece éste y todos los demás métodos que describiremos. Así **RuleBase.match()** toma un simple parámetro booleano y se desplaza a través de la lista de reglas ruleList. Sí el parámetro "*test*" es verdadero, llama al método **Rule.check()** para probar todas las cláusulas antecedentes y determinar el valor Truth de la regla. Sí el parámetro test es falso, el método **match()** simplemente mira el valor Truth actual de la regla, sí éste es verdadero, y la regla no ha sido disparada, esta regla es agregada en un vector ***matchList***[]; y sí el valor de la regla es false se salta esta regla y continua con las siguientes en la lista.

El método **RuleBase.selectRule()**, toma el conjunto de reglas en conflicto (que es un vector de reglas) como único parámetro de entrada, la implementación es bastante simple, usa especificidad [2], que consiste en utilizar el número de cláusulas antecedentes o premisas como elemento primario para seleccionar cuál regla vamos a disparar. Sí dos o más reglas tienen el mismo número de premisas, se seleccionan en orden de entrada es decir la primera encontrada, esta selección trabaja de la siguiente forma. Comenzamos tomando la primera regla del vector, la cual es designada como nuestra mejor regla (*bestRule*), también tomamos el número de premisas o cláusulas antecedentes como el valor máximo o mejor valor. En el ciclo, recorremos el resto de las siguientes reglas, y sí encontramos una regla que tenga más premisas que la previa, ponemos a dicha regla como nuestra mejor (*bestRule*) y su correspondiente número de premisas como valor máximo, después de buscar en todas las reglas de la lista se regresará la mejor regla a ser disparada, es decir la que contiene mayor número de premisas.

El método **Rule.check()**, es usado durante el proceso de encadenamiento hacia adelante para probar las cláusulas antecedentes de la regla. Sí cualquiera de las cláusulas tiene un valor Truth indefinido, entonces este método regresa un valor nulo. Sí cualquiera de las cláusulas es falsa, entonces el valor Truth de la regla se pone en *false* y también un valor *false* es regresado por este método. Sí todas las cláusulas de las antecedentes de la regla son verdaderas, el valor Truth de la regla es también colocado en verdadero, y también un valor Truth es regresado por este método.

El método **Rule.fire(),** es usado durante el proceso de encadenamiento hacia adelante, cuando una regla con un valor Truth igual a verdadero es seleccionada para ser disparada; cuando una regla es disparada, la variable booleana "*fired*" de la clase **Rule**, se le asigna el valor verdadero para indicar que la regla ha sido disparada y la variable de regla del lado izquierdo de la cláusula consecuente, le es asignado el valor especificado en su lado derecho.

El método **checkRules(),** es entonces llamado para reexaminar aquellas reglas que hacen referencia a la variable de la cláusula consecuente, esto ocasiona que a las demás cláusulas que tienen a la variable de regla se les asigne este valor.

El método **Rule.checkRules()**, reexamina cada cláusula que hace referencia a la variable de regla que fue cambiada por la activación de una regla (que fue disparada). Esto es, porque ahora la variable de regla tiene un nuevo valor y todas las cláusulas antecedentes que hacen referencia a ella y que tienen un valor indefinido o nulo, se les puede poner un valor Truth igual a verdadero o falso. Esto significa, que las reglas que hacen referencia a aquellas cláusulas pueden ser también evaluadas.

4.2.2.8 Encadenamiento hacia atrás.

El método **RuleBase.BackwardChain(),** es aplicado durante el proceso de inferencia, este método toma como único parámetro una cadena que es el nombre de la variable meta. Éste nombre de variable es usado para obtener la variable de regla (*RuleVariable*) de una lista de objetos. Todas las cláusulas que hacen referencia a la variable meta son enumeradas y se utiliza entonces un ciclo para procesar cada objeto de tipo *Clause.* Si esta no es una cláusula consecuente, es ignorado y continuamos a través del ciclo para examinar la siguiente cláusula, sí si lo es, se coloca en una pila de metas (*goalClauseStack),* y es entonces, cuando tenemos una referencia de la regla que contiene a dicha cláusula como consecuente. A continuación llamamos al método **Rule.BackChain()** sobre esta regla, y por medio de esta llamada podemos comprobar si la regla es verdadera ó no. El método **Rule.BackChain(),** hace llamadas recursivas a **RuleBase.backwardChain()** sí es necesario, para seguir una cadena de inferencias a través de la base de reglas y con el fin de encontrar sí la meta original *goalClause* es verdadera ó falsa.

El método **Rule.backChain()** regresará un valor para Truth (verdadero o falso) de la regla que analiza, si se cumplen las siguientes condiciones:

- Sí el valor Truth de la regla es nulo, no podemos determinar si la meta actual goalClause es verdadera ó falsa. Tampoco si la base de reglas está incompleta, o si el usuario provee valores no validos cuando estos se le solicitan.
- Sí la regla fue probada verdadera, disparamos la regla asignando a la variable meta actual, el valor del lado derecho de la cláusula a la que pertenece (*goalClause*) también agregamos una referencia a la variable para decirle que

la regla produce ese valor, luego retiramos a la cláusula de la pila de metas (*goalClauseStack*) y desplegamos un mensaje de éxito. Si la pila de metas esta vacía, hemos terminado el ciclo de encadenamiento hacia atrás y desplegamos un mensaje de fin del ciclo de inferencia, con lo cual también terminamos con el proceso de inferencia.

- Sí la regla fue falsa, extraemos la meta actual (goalClause) de la pila de metas y desplegamos un mensaje de falla, continuamos a través del ciclo de inferencia para pasar a la siguiente meta.

El método **Rule.backChain(),** tratará siempre de probar si una regla es verdadera o falsa por medio de llamadas recursivas al método **RuleBase.backwardChain()** hasta que un valor Truth para dicha regla pueda ser determinado.

Este método consiste en un ciclo del tipo for() [6][13], donde cada cláusula antecedente de una regla es evaluada. Sí la variable de una cláusula antecedente es indefinida, entonces el método **RuleBase.backwardChain()** es llamado para determinar su valor. Sí el valor no puede ser inferido, se le pregunta al usuario mediante el método **RuleVariable.askUser()**. Una vez que el usuario proporciona un valor, la cláusula es probada usando el método **Clause.check()**, sí la cláusula es verdadera, continuamos con el ciclo para evaluar las siguientes cláusulas; sí la cláusula fue falsa, se sale del ciclo for() reportando que el valor Truth de la regla es falso ya que una de las cláusulas antecedentes es falsa.

Si a través del ciclo obtenemos que todas las cláusulas antecedentes son verdaderas, entonces regresamos un valor Truth=verdadero para la regla.

El proceso de operación de estos métodos, se muestra en la figura 4.5.

4.2.3 Implementación de la Interfaz de usuario.

A continuación se describe la implementación de la interfaz de usuario para el ambiente de desarrollo de los sistemas expertos, esta interfaz, fue desarrollada con una herramienta visual para programar en Java (JBuilder2) [6]. Con esta interfaz de usuario, podemos capturar la base de conocimiento y representarlo mediante reglas, además podemos aplicar los diferentes procesos de inferencia para probar la base de conocimiento, y obtener resultados, realizando así las diferentes etapas del desarrollo de sistemas expertos basados en reglas. Para cada una de las etapas que se llevan a cabo en el desarrollo de un sistema experto, se tienen diferentes pantallas en las que se hacen diferentes, estas pantallas son las siguientes:

Definición de Variables y sus valores.
Construcción de cláusulas e implementación de reglas.
Procesos de inferencia.
Manejo de Bases de conocimiento.
Opciones.

A continuación se detalla la función de cada una de las pantallas, para más información acerca del uso de la interface, el apéndice B proporciona el instructivo

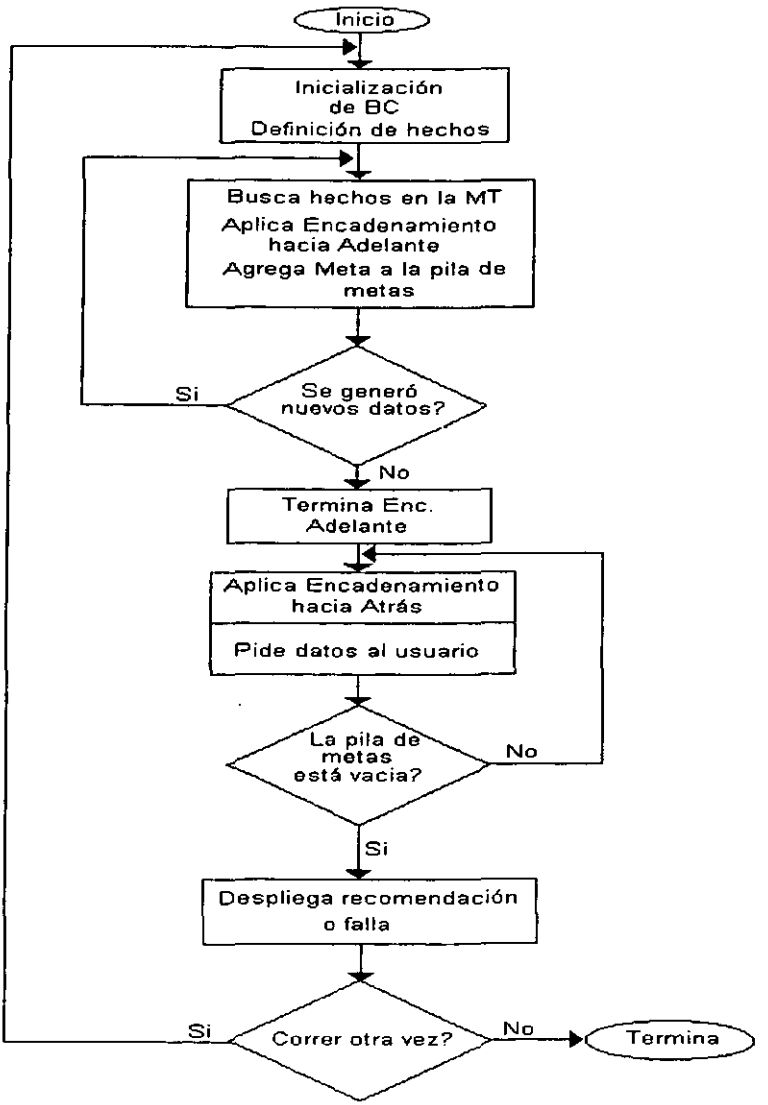


Figura 4.5 Proceso de Inferencia sobre la base de conocimiento

completo de operación para el desarrollo de sistemas expertos basados en reglas usando esta interfaz, en esta sección únicamente describimos la función de cada una de las pantallas.

Definición de variables.

En esta pantalla, se pueden definir las variables del sistema experto, estas variables serán utilizadas en la definición de las cláusulas y estas a su vez para la implementación de las reglas; también se pueden accesar durante los procesos de inferencia para establecer condiciones iniciales, es decir establecer los hechos de la base de conocimiento.

Para cada variable se definen los siguientes parámetros: *Nombre* que es una cadena alfanumérica y no debe empezar con número, una cadena alfanumérica *prompt* que es el enunciado para la pregunta utilizada al solicitar al usuario información acerca de la variable durante los procesos de inferencia, y las opciones de la variable, que son todos los valores permitidos que la variable puede tomar durante los procesos de inferencia y cuando se le solicita al usuario. La figura 4.6 nos muestra esta pantalla.

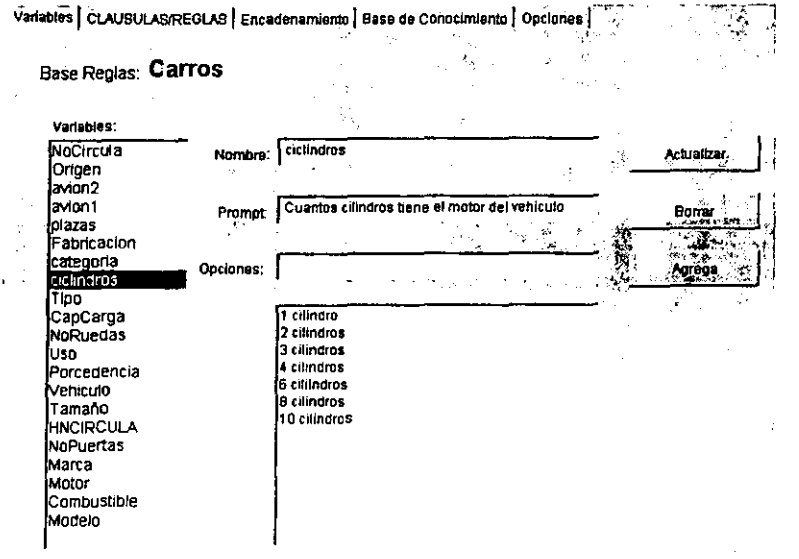


Figura 4.6 Definición de variables.

En el lado izquierdo superior, se despliega el nombre de la base de reglas con la que se esta trabajando, una lista al lado izquierdo nos permite ver todas las variables que se han definido y que forman parte de la base de conocimiento; con los tres botones de control podemos: Definir nuevas variables, modificar las ya

existentes y borrar los campos de captura, un mensaje de alerta avisara la situación en caso de haber algún problema. Esta pantalla, cuenta con procedimientos de seguridad para evitar declarar variables con el mismo nombre y también cuando falta información por capturar. Cuando se define una nueva variable, ésta es inmediatamente agregada a la base de conocimiento, y esta lista para ser integrada dentro de una cláusula y ésta a su vez para serlo de una regla.

Esta pantalla trabaja en conjunto con las clases (Programas e Java) que sirven para la definición de variables y su integración a la base de conocimiento, también actualiza los datos nuevos en otras pantallas de la interfaz para que puedan ser utilizados en sus procedimientos.

Construcción de cláusulas e implementación de reglas.

En esta pantalla, se realizan dos tareas: (1) construcción de Cláusulas con las variables que ya han sido definidas y (2) con estas cláusulas, se implementan las reglas necesarias para así completar la base de conocimiento del sistema experto.

Para construir un objeto de la clase **Clause**, se necesitan tres parámetros de una variable: Nombre de la variable, una condición de correspondencia, y un valor determinado para dicha variable. Se puede seleccionar cualquier variable y a ésta cualquiera de sus valores permitidos, esta pantalla cuenta con procedimientos de seguridad para evitar errores durante la construcción de las cláusulas y por consiguiente de las reglas. Debemos recordar que la implementación de las reglas del sistema experto, constituye parte de la base de conocimiento, y tal vez la más importante, por lo cual no deben tener datos erróneos.

Las reglas que se implementan (del tipo Si-Ent.) deben tener un nombre para identificarlas en la base de conocimiento, pueden tener hasta 10 premisas o cláusulas antecedentes en la parte Si; todas las premisas, están unidas por una conjunción, y sólo se permite una conclusión o cláusula consecuente en la parte Ent.

Las casillas de selección en la parte superior, permiten construir las cláusulas sólo con datos válidos estos son: Variables y sus valores permitidos; con el botón de control "Y" hacemos la conjunción de las premisas, al realizar esto se van desplegando cada una en una lista de cláusulas.

En las casillas de selección de en medio, se define la conclusión o consecuente, también solo con datos de variables válidos.

Esta pantalla, cuenta con procesos de seguridad para evitar errores en la definición de los valores que se le asignan a una variable con la que se construye una cláusula, también verifica que no se repitan nombres de las reglas.

Al terminar una regla, esta se agrega a la base de conocimiento; dos campos en la parte inferior de la pantalla, permiten al usuario ver la estructura de todas las

reglas que ya han sido integradas a la base de conocimiento y en su caso hacer modificaciones. Cuando una nueva regla ha sido definida, ésta entra inmediatamente a la base de conocimiento (en nuestro caso la hemos definido como Base de Reglas) y se puede utilizar en el siguiente ciclo de inferencia para obtener resultados, de igual manera sucede cuando una regla ya existente ha sido modificada.

Los procedimientos de control de esta pantalla, trabajan en conjunto con las clases que definen a las cláusulas y las reglas descritas en sección anterior. La figura 4.6 nos muestra esta pantalla.

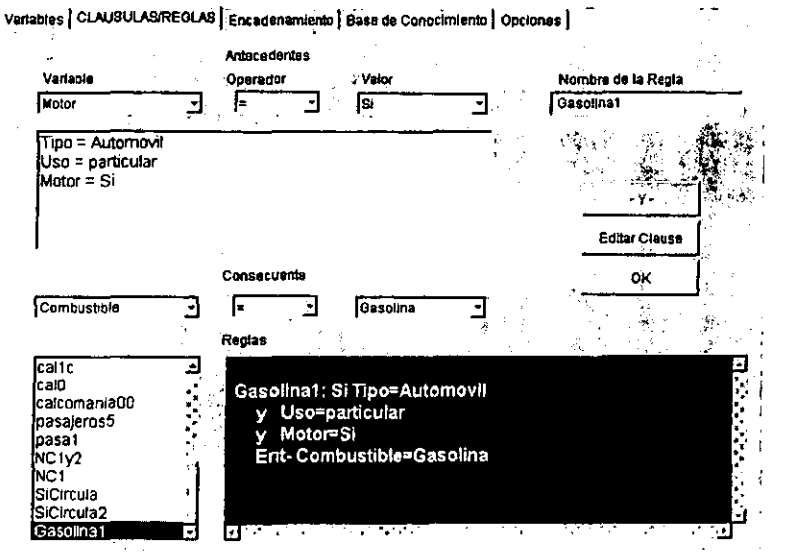


Figura 4.6 Definición de Cláusulas y Reglas.

Procesos de Inferencia.

Esta es la pantalla de pruebas para la base de conocimiento. Una vez que se han terminado de implementar las reglas, o inclusive antes de terminar, se puede utilizar esta pantalla para probar todas las reglas que hasta el momento se han implementado, de hecho esta es una tarea muy común durante el desarrollo de los sistemas expertos.

Este es el momento ideal para comentar que este sistema nos permite estar trabajando con diferentes sistemas expertos, la casilla de selección en la parte superior izquierda, permite seleccionar la base de reglas a probar. Una vez seleccionada la base de reglas, mediante los botones de control, se pueden hacer las siguientes funciones:

- Ejecutar alguno de los procesos de inferencia: encadenamiento hacia adelante o encadenamiento hacia atrás.
- Reinicializar la base de conocimiento, esto simplemente es asignar a todas las variables un valor nulo.
- Establecer condiciones iniciales o hechos, esto es, asignar a ciertas variables un valor permitido diferente a nulo.
- Establecer cuál será la meta a buscar durante el proceso de encadenamiento hacia atrás.

Con las casillas de selección en la parte superior, podemos establecer condiciones iniciales o hechos, esto es, asignar a determinadas variables un valor diferente de nulo (siendo alguno de los permitidos para dicha variable) como sabemos esto es muy importante para pruebas de la base de reglas, sobre todo cuando se trabaja con encadenamiento hacia adelante.

Una casilla mas de selección en la parte central derecha, permite definir una meta a determinar, y así probar la base de reglas cuando se trabaja con encadenamiento hacia atrás. La figura 4.8 nos muestra esta pantalla.

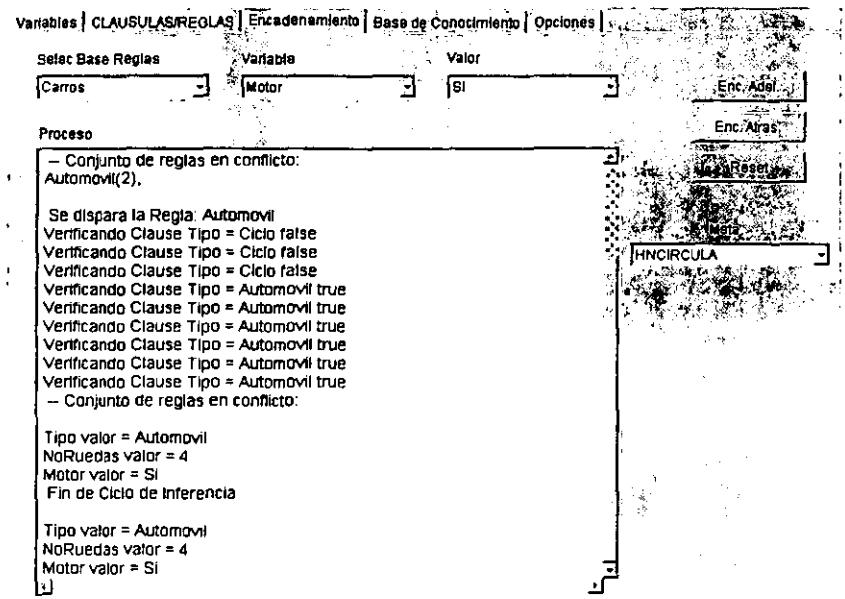


Figura 4.8 Pruebas de la base de conocimiento.

Un campo de despliegue en la parte central, permite ver los resultados de los procesos de inferencia que son aplicados, esto constituye el sistema de explicación para dichos procesos de inferencia. En este campo se despliegan:

- Los valores de las variables al inicio y fin del ciclo de inferencia.
- El conjunto de reglas en conflicto y la resolución de éste, seguido de la selección y prueba de cada una de las reglas en conflicto.
- Las reglas que se disparan, cuando en el ciclo de inferencia se han probado sus premisas y se ha comprobado que son verdaderas, también se despliegan los nuevos datos que son inferidos a través de la activación de estas reglas.
- En caso de que se disparen reglas que tienen recomendaciones en la parte de la conclusión, en este campo se despliegan las recomendaciones que da el sistema

Hay que recordar que estamos en el ambiente de desarrollo, una vez que se han terminado las pruebas, el sistema experto será formateado en un programa que corre a través de la Web, y la parte de explicación no es tan detallada como aquí.

Manejo de Bases de conocimiento.

En esta pantalla, se realiza la administración de los sistemas expertos que se implementan, una vez que se han terminado todas las pruebas pertinentes, la base de conocimiento (base de reglas, variables y otros parámetros) se almacena en un archivo que puede ser transferido al servidor Web. En el servidor Web, hay otros programas que toman la base de conocimiento y le aplican los mismos procesos de inferencia para que el sistema experto pueda ejecutarse en páginas Web, este proceso se describe con más detalle en la ultima sección de este capítulo.

Con esta pantalla, podemos guardar la base de reglas, también podemos recuperar los archivos de la base de reglas, para modificaciones y volver a guardarlos.

El ambiente de desarrollo permite tener almacenado en la memoria de trabajo, diferentes bases de reglas, y seleccionar con la que queramos trabajar, varios campos de despliegue muestran la estructura de la base de conocimiento seleccionada. En la figura 4.8 se muestra esta pantalla.

Una vez que se ha probado la base de reglas y cumple con los requerimientos, está lista para ser montada en la Web, para esto en la carpeta de opciones, con el botón de control "Genera Applet" el usuario seleccionará el directorio del servidor Web donde se colocarán los archivos necesarios para que la nueva base de reglas corra en la Web. La pantalla de opciones, se muestra en la figura 4.9

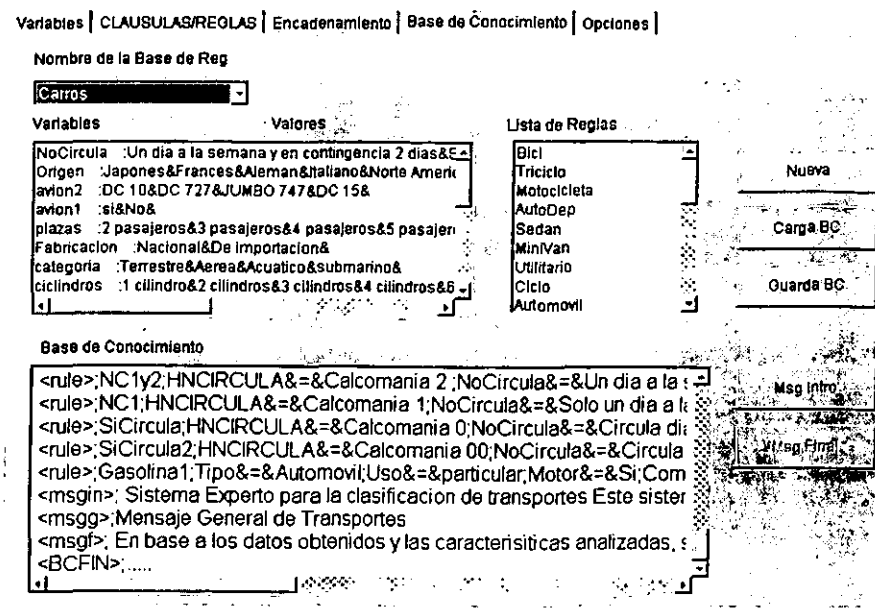


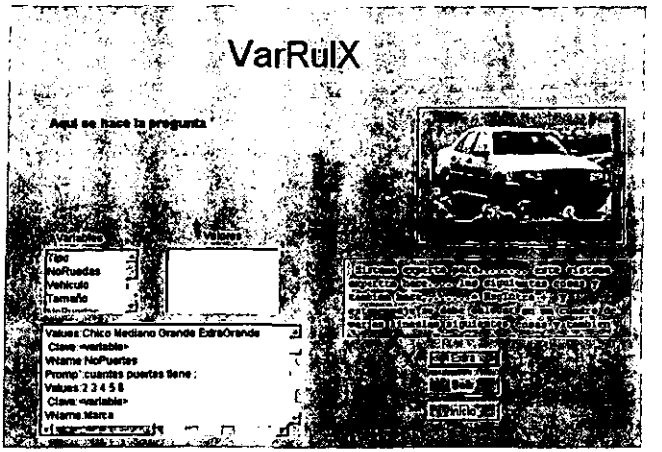
Figura 4.9 Manejo de bases de reglas.

La presentación final del sistema experto, es muy similar a lo que se ve en el ambiente de desarrollo, pero los sistemas expertos basados en Applets Java, se ejecutan dentro de formas html que son llamados al servidor Web por medio de un navegador de Internet.

Cada Applet Java que se genera, tiene integrada tanto la base de conocimiento, como su máquina de inferencia, por lo tanto antes de correrlo, el usuario debe descargar el Applet completo, una vez que esto se logra, el Applet se inicializa automáticamente y así el sistema experto empieza a trabajar.

Dentro del ambiente de desarrollo, se establecen varios formatos para la presentación final, una característica muy importante de los Applets Java, es que con pequeñas modificaciones en el código fuente, estos programas además de que pueden correr tanto en la Web, también pueden ejecutarse como aplicaciones independientes utilizando la Máquina Virtual de Java. Algunos de los formatos que se incluyeron en esta aplicación tienen esta cualidad ya que para fines de evaluación, los usuarios finales, podrán descargarlos como archivos comprimidos e instalarlos en sus computadoras.

Un ejemplo de la presentación final de un sistema experto corriendo en la Web se muestra en la figura 4.10.



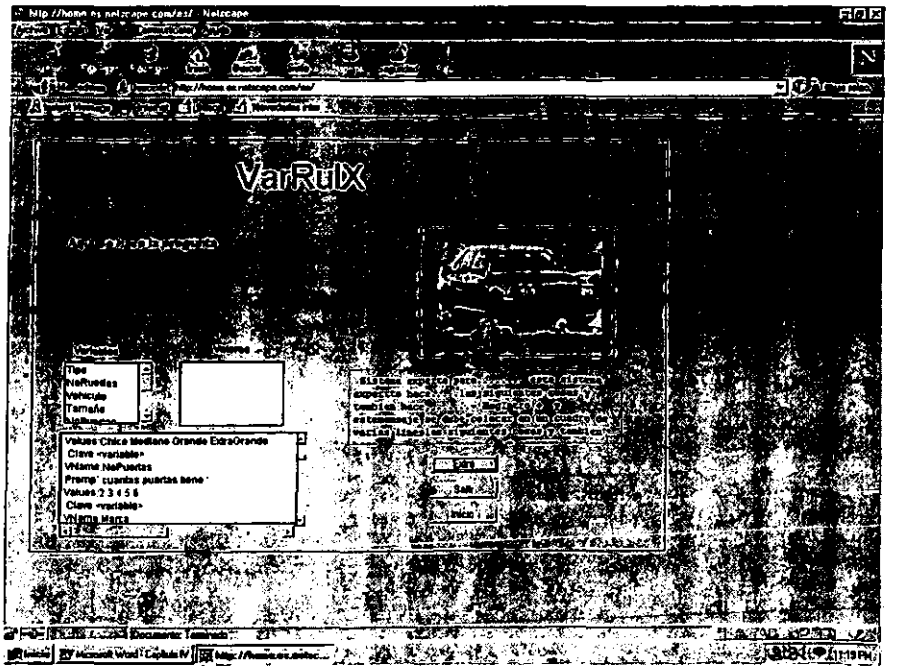


Figura 4.10 Un Applet Java con sistema experto.

Resumen.

Hay varias alternativas para la ejecución de programas del lado del cliente. Java ha sido la más popular y exitosa, ya que un marco de trabajo para crear Applets, que se ejecuten en la maquina del cliente. El Applet Java, tiene acceso a la Web, puede cargar cualquier archivo de datos e imágenes requeridas por el sistema experto. Además de resolver muchos de los problemas que se presentan al correr en un servidor, hacen que el mantenimiento del sistema sea más fácil.

Con Java podemos hacer Applets que corren mediante la Máquina Virtual de Java (JVM) en la computadora cliente. La JVM convierte el programa Java en operaciones que se pueden ejecutar en diferentes plataformas, la misma JVM da a Java su portabilidad para muchos tipos de computadoras. La consideración más grande a tomar en cuenta de los Applets Java, es su tamaño.

Podemos diseñar un sistema experto basado en un Applet, pero este puede ser bastante complicado e inaceptable para descargarse, a menos que seamos muy cuidadosos en su diseño.

Una de las mejores ventajas de los Applets Java es que son muy escalables, el servidor solo envía el archivo del Applet al cliente; actualmente sabemos que descargar archivos es algo que los servidores hacen muy bien. Aunque el número de usuarios se incremente, todo el procesamiento se lleva acabo en las máquinas cliente inclusive los sistemas altamente interactivos, no proporcionan carga adicional al servidor.

El Applet se comunica con el servidor para los datos, o los datos le son pasados al al inicializarse, pero la interacción con el usuario se lleva a cabo en la computadora cliente, inclusive también se pueden comunicar con otros Applets para obtener datos.

Conclusiones.

Con este trabajo se logró el objetivo, la arquitectura planteada se implementó, y se realizaron pruebas para comprobar la operación correcta y adecuada. Cada uno de los elementos necesarios, fue configurado, y se desarrollaron las aplicaciones necesarias para la operación del sistema.

Podemos desarrollar sistemas expertos para correr en la Web, como programas que se insertan en páginas html y que tienen tanto la base de conocimiento como la máquina de inferencia integradas en un solo programa.

La característica más fascinante y poderosa de los sistemas expertos, que los distingue de la mayoría de las aplicaciones tradicionales de la computación, es su capacidad para enfrentar problemas que constituyen un reto al mundo real, por medio de la aplicación de procesos que reflejan el discernimiento y la intuición humana.

Implementar sistemas expertos por medio de Applets Java, es relativamente nuevo, pero con el desarrollo de este trabajo, observamos que proporciona elementos interactivos que pueden ser fácilmente integrados y actualizados en las páginas html.

Los sistemas expertos basados en Applets, proveerán recomendaciones para la solución de problemas específicos como selección de un producto, soporte técnico en alguna área específica. Estos pueden ser, desde un simple letrero que contesta preguntas, hasta una configuración muy compleja para la toma de decisiones.

Los sistemas expertos se están empleando en una amplia variedad de aplicaciones que comprenden entre otras, diagnóstico, planeación, predicción, diseño, interpretación, control, monitoreo de estado e instrucción. Y en el futuro, a medida que se produzcan y desarrollen nuevas arquitecturas de equipos, que soporten de una manera más directa la ejecución de sistemas expertos y se perfeccionen la tecnología de Inteligencia Artificial, es razonable esperar un desarrollo de sistemas que se aproximen asintoticamente al comportamiento humano. El desarrollo de tales sistemas, nos permitirá no solamente ofrecer opciones técnicas más potentes sino también alimentar más nuestro propio conocimiento acerca del proceso del pensamiento humano.

Puesto que Java proporciona un completo acceso a la Web, las recomendaciones de los sistemas expertos pueden ser ligadas e integradas en todos los sitios Web.

Los sistemas expertos basados en Applets, proporcionarán elementos interactivos para un diseño de pagina que serán fácilmente integrados y actualizados.

Montar sistemas en la Web por medio de Applets, es relativamente nuevo y crece rápidamente y esta alternativa puede convertirse en la principal técnica del futuro para montar en la Web el conocimiento de los expertos.

Bibliografía

[1] Peter Jackson Introductión to Expert Systems.
Adison-Wesley publishing Company 1990

[2] David W. Rolston Principles of Artificial Intelligence and Expert Systems.
McGaw Hill 1991

[3] Hayes Roth Waterman Building Expert Systems.

[4] Joseph P. Bigus and Jennifer Bigus Contructing Intelligent Agents with Java.
Wiley Computer publishing 1997.

[5] E. Rich Artificial Intelligence.
McGraw Hill Inc.1983.

[6] Donald Doherthy y Michelle M. Manning. Manual de programación Borland.
Jbuilder3. Prentice Hall Hispanoamericana S.A. México 2000.

[7] Stuart J. Russell and Peter Norving. Inteligencia Artificial un enfoque moderno.
Prentice Hall Hispanoamericana S.A. México 1995.

[8] Efraim Turban Expert Systems and Applied Artificial Intelligence.
California State University at long beach 1992.

[9] John Turkin Expert Systems Design and Development

[10] Sidnie Feil TCP/IP Arquitectura Protocolos e Implementación.

[11] DG. Dologite. Constructing Expert Systems using Vpexpert.
MacMillan publishing company 1993.

[12] Mike Cohn, Bryan Morgan, Michael Morrison, Michael T. Nygard, Dan Joshi and
Tom Trinco .WWW.MCP/SAMS Java Developer's Reference
SAMS-NET 1995.

[13] Laura Lemay Charles R. Perkins Aprendiendo Java 2.
Prentice Hall Hispanoamericana SA de CV 1996.

[14] Tomas Dean and James Allen Artificial Intelligence Theory and Practice.
Benjamin/Cummings publishers Inc. 1995

[15] Forgy y C.L. (1982) Artificial Intelligence. Rete: as a fast algorithm for the many
pattern/many object pattern match problem.

[16] PCAI Where the Intelligent Technology Meets the Real
World.November/December 2000 Volume 4 number 6 .

Código Fuente

```
/*Clase Variable, es la clase principal para la construcción y manejo de
variables que sirven para formar las cláusulas y estas a su vez para
formar las reglas, tiene métodos que se encargan del control de las variables
declaradas
*/
import java.util.*;
import java.io.*;
public abstract class Variable {
public String name ;
String value;
int column ;
public Vector labels;
public Vector Images;
Variable() {} ;
Variable(String Name) {name = Name; value = null; }
void  setValue(String val) { value = val ; }
String getValue() { return value; }
// Metodo setLabels, usado para establecer los valores validos de una Variable
public void setLabels(String Labels) {
   labels = new Vector() ;
   StringTokenizer tok = new StringTokenizer(Labels,"&");
   while (tok.hasMoreTokens()) {
      labels.addElement(new String(tok.nextToken()));
     // Images.addElement(new image( nombre de la imagen nula);
   }
}
// Metodo getLabel(), regresa el valor del indice especificado
String getLabel(int index) {
   return (String)labels.elementAt(index);}

   // Metodo getLabels, regresa una cadena que contiene todos los valores
public String getLabels() {
   String labelList = new String();
   Enumeration enum = labels.elements() ;
   while(enum.hasMoreElements()) {
      labelList += enum.nextElement() + "&" ;}
   return labelList ;
}
// Dado un label, regresa su indice
int getIndex(String label) {
   int i = 0, index = 0  ;
   Enumeration enum = labels.elements() ;
   while(enum.hasMoreElements()) {
     if (label.equals(enum.nextElement()))
     { index = i ; break ; }
     i++;}
   return i;
}
boolean categorical() {
  if (labels != null) {
    return true ;
  } else {
    return false ;}
}
```

```
// Metodo setColumn(), usado por la clase DataSet
public void setColumn(int col) { column = col ; }
public abstract void computeStatistics(String inValue) ;
public abstract int normalize(String inValue, float[] outArray, int inx);
public int normalizedSize() { return 1 ; }
public String getDecodedValue(float[] act, int index) { return String.valueOf(act[index]); }

}
```

/* Clase RuleVariable, esta clase hereda de la superclase Variable, y sirve para construir las variables de las reglas, estas junto con las reglas, son parte de la base de conocimiento de un sistema experto
***/**

```
import java.util.*;
import java.awt.* ;
public class RuleVariable extends Variable {
Frame frame = new Frame("Variable desconocida");
Vector clauseRefs ;  // clauses a las que hace referencia esta variable
public RuleVariable(String Name) {
      super(Name);
      clauseRefs = new Vector();}
public void setValue(String val) { value = val;
                  updateClauses(); }
void addClauseRef(Clause ref) { clauseRefs.addElement(ref) ; }
void updateClauses() {
  Enumeration enum = clauseRefs.elements() ;
  while(enum.hasMoreElements()) {
   ((Clause)enum.nextElement()).check() ; // checa la condicion de truth
  }
}
public String promptText ;  // usado para solicitar al usuario el valor de la variable
String ruleName ;          // si el valor es inferido, null = valor dado
void   setRuleName(String rname) { ruleName = rname; }
public void   setPromptText(String text) { promptText = text; }

// Estos métodos no son usados en Variables de Regla (Rule Variables)
public void computeStatistics(String inValue){} ;
public int normalize(String inValue, float[] outArray, int inx) {return inx;}
};
```

/*Clase Clause, sirve para generar las cláusulas utilizadas en las reglas, las cláusulas se declaran según la norma Objeto-Atributo-Valor
***/**

```
import java.util.*;
import java.io.*;
public class Clause {
   Vector ruleRefs ;
public   RuleVariable lhs ;
   String rhs ;
   Condition  cond ;
   Boolean consequent ;  // true o false
   Boolean truth ;   // Estado = null(desconocido), true o false

public Clause(){}
public Clause(RuleVariable Lhs, Condition Cond, String Rhs)
```

```
    {
     lhs = Lhs ; cond = Cond ; rhs = Rhs ;
     lhs.addClauseRef(this) ;
     ruleRefs = new Vector() ;
     truth = null ;
     consequent = new Boolean(false);}

  public Clause(RuleVariable LHS,String Cadena){
    StringTokenizer ClauseX = new StringTokenizer(Cadena," ");
    lhs =LHS;
    lhs.addClauseRef(this);
    String Alfa= ClauseX.nextToken();
    cond= new Condition(ClauseX.nextToken());
    rhs=ClauseX.nextToken();
    ruleRefs = new Vector();
    truth = null;
    consequent = new Boolean(false);}
  void addRuleRef(Rule ref) { ruleRefs.addElement(ref) ; }
  Boolean check() {
       if (consequent.booleanValue() == true) return null ;
       if (lhs.value == null) {
          return truth = null ;  //No puede checar la variable, su valor es indefinido
       } else {
       switch(cond.index) {
       case 1: truth = new Boolean(lhs.value.equals(rhs)) ;
        RuleBase.appendText("\nProbando Clause " + lhs.name + " = " + rhs + " " + truth);
        break ;
       case 2: truth = new Boolean(lhs.value.compareTo(rhs) > 0) ;
        RuleBase.appendText("\nProbando Clause " + lhs.name + " > " + rhs + " " + truth);
        break ;
       case 3: truth = new Boolean(lhs.value.compareTo(rhs) < 0) ;
        RuleBase.appendText("\nProbando Clause " + lhs.name + " < " + rhs + " " + truth);
        break ;
       case 4: truth = new Boolean(lhs.value.compareTo(rhs) != 0) ;
        RuleBase.appendText("\nProbando Clause " + lhs.name + " != " + rhs + " " + truth);
        break ;
       }
       return truth ;
       }
    }
  void isConsequent() { consequent = new Boolean(true); }
    Rule getRule() { if (consequent.booleanValue() == true)
       return (Rule)ruleRefs.firstElement() ;
       else return null ;}
};
```

/* Clase Condition, sirve para establecer los atributos que son utilizados en las cláusulas, y que determinana el patron de comparación durante los procesos de inferencia
***/**

```
import java.util.*;
public class Condition {
  int index ;
  String symbol ;
 public  Condition(String Symbol) {
```

```
      symbol = Symbol ;
      if (Symbol.equals("=")) index = 1 ;
      else if (Symbol.equals(">")) index = 2 ;
      else if (Symbol.equals("<")) index = 3 ;
      else if (Symbol.equals("!=")) index = 4 ;
      else index = -1 ;}
   String asString() {
      String temp = new String() ;
      switch (index) {
       case 1: temp = "=" ;
       break;
       case 2: temp = ">" ;
       break ;
       case 3: temp = "<" ;
       break;
       case 4: temp = "!=" ;
       break;
      }
    return temp ;
   }
   String asStringX(){
    String temp2 = new String();
    switch(index) {
      case 1: temp2="Igual";
      break;
      case 2: temp2="Mayorque";
      break;
      case 3: temp2="Menorque";
      break;
      case 4: temp2="Difer";
      break;
    }
    return temp2;
   }
}
```

/*Clase Rule, sirve para construir las reglas de la Base de conocimiento en base a todas las posibles cláusulas que se pueden hacer por medio de las variables de reglas, el numero de reglas que se pueden hacer, esta en proporción al numero de variables de regla declaradas.
***/**

```
import java.util.*;
import java.io.*;
import java.awt.* ;
import Dialog1;
public class Rule {
  RuleBase rb ;
public  String name ;
  Clause antecedents[] ;       //Permite hasta 10 antecedentes por Regla y
public  Clause consequent ;    //solo se permite 1 consecuente.
  Boolean truth;               //Estados = (null=nulo, true, y false).
  boolean fired=false;
  Dialog1 BOX2;
  Frame frame = new Frame("Variable desconocida");
//Constructores de la Regla, se permiten reglas hasta con 10 premisas.
```

```
  public  Rule(RuleBase Rb, String Name, Clause lhs, Clause rhs) {
     rb = Rb ;
     name = Name ;
     antecedents = new Clause[1] ;
     antecedents[0] = lhs ;
     lhs.addRuleRef(this) ;
     consequent = rhs ;
     rhs.addRuleRef(this) ;
     rhs.isConsequent() ;
     rb.ruleList.addElement(this) ;  // se agrega a la lista de Reglas
     truth = null ;}
  public Rule(RuleBase Rb, String Name, Clause lhs1, Clause lhs2, Clause rhs) {
     rb = Rb ;
     name = Name ;
     antecedents = new Clause[2] ;
     antecedents[0] = lhs1 ;
     lhs1.addRuleRef(this) ;
     antecedents[1] = lhs2 ;
     lhs2.addRuleRef(this) ;
     consequent = rhs ;
     rhs.addRuleRef(this) ;
     rhs.isConsequent() ;
     rb.ruleList.addElement(this) ;  //  se agrega a la lista de Reglas
     truth = null ;}
  public Rule(RuleBase Rb, String Name, Clause lhs1, Clause lhs2,
                           Clause lhs3, Clause rhs) {
     rb = Rb ;
     name = Name ;
     antecedents = new Clause[3] ;
     antecedents[0] = lhs1 ;
     lhs1.addRuleRef(this) ;
     antecedents[1] = lhs2 ;
     lhs2.addRuleRef(this) ;
     antecedents[2] = lhs3 ;
     lhs3.addRuleRef(this) ;
     consequent = rhs ;
     rhs.addRuleRef(this) ;
     rhs.isConsequent() ;
     rb.ruleList.addElement(this) ;  // se agrega a la lista de Reglas
     truth = null ;}
  public Rule(RuleBase Rb, String Name, Clause lhs1, Clause lhs2,
                           Clause lhs3, Clause lhs4,
                           Clause rhs) {
     rb = Rb ;
     name = Name ;
     antecedents = new Clause[4] ;
     antecedents[0] = lhs1 ;
     lhs1.addRuleRef(this) ;
     antecedents[1] = lhs2 ;
     lhs2.addRuleRef(this) ;
     antecedents[2] = lhs3 ;
     lhs3.addRuleRef(this) ;
     antecedents[3] = lhs4 ;
     lhs4.addRuleRef(this) ;
     consequent = rhs ;
     rhs.addRuleRef(this) ;
```

```
    rhs.isConsequent() ;
    rb.ruleList.addElement(this) ; // se agrega a la lista de Reglas
    truth = null ;}
  public Rule(RuleBase Rb, String Name, Clause lhs1, Clause lhs2,
                           Clause lhs3, Clause lhs4,
                           Clause lhs5,
                           Clause rhs) {
    rb = Rb ;
    name = Name ;
    antecedents = new Clause[5] ;
    antecedents[0] = lhs1 ;
    lhs1.addRuleRef(this) ;
    antecedents[1] = lhs2 ;
    lhs2.addRuleRef(this) ;
    antecedents[2] = lhs3 ;
    lhs3.addRuleRef(this) ;
    antecedents[3] = lhs4 ;
    lhs4.addRuleRef(this) ;
    antecedents[4] = lhs5 ;
    lhs5.addRuleRef(this) ;
    consequent = rhs ;
    rhs.addRuleRef(this) ;
    rhs.isConsequent() ;
    rb.ruleList.addElement(this) ; // se agrega a la lista de Reglas
    truth = null ;}
  public Rule(RuleBase Rb, String Name, Clause lhs1, Clause lhs2,
                           Clause lhs3, Clause lhs4,
                           Clause lhs5, Clause lhs6,
                           Clause rhs) {
    rb = Rb ;
    name = Name ;
    antecedents = new Clause[6] ;
    antecedents[0] = lhs1 ;
    lhs1.addRuleRef(this) ;
    antecedents[1] = lhs2 ;
    lhs2.addRuleRef(this) ;
    antecedents[2] = lhs3 ;
    lhs3.addRuleRef(this) ;
    antecedents[3] = lhs4 ;
    lhs4.addRuleRef(this) ;
    antecedents[4] = lhs5 ;
    lhs5.addRuleRef(this) ;
    antecedents[5] = lhs6 ;
    lhs6.addRuleRef(this) ;
    consequent = rhs ;
    rhs.addRuleRef(this) ;
    rhs.isConsequent() ;
    rb.ruleList.addElement(this) ; // se agrega a la lista de Reglas
    truth = null ;
  public Rule(RuleBase Rb, String Name, Clause lhs1, Clause lhs2,
                           Clause lhs3, Clause lhs4,
                           Clause lhs5, Clause lhs6,
                           Clause lhs7,
                           Clause rhs) {
    rb = Rb ;
    name = Name ;
```

```
    antecedents = new Clause[7] ;
    antecedents[0] = lhs1 ;
    lhs1.addRuleRef(this) ;
    antecedents[1] = lhs2 ;
    lhs2.addRuleRef(this) ;
    antecedents[2] = lhs3 ;
    lhs3.addRuleRef(this) ;
    antecedents[3] = lhs4 ;
    lhs4.addRuleRef(this) ;
    antecedents[4] = lhs5 ;
    lhs5.addRuleRef(this) ;
    antecedents[5] = lhs6 ;
    lhs6.addRuleRef(this) ;
    antecedents[6] = lhs7 ;
    lhs7.addRuleRef(this) ;
    consequent = rhs ;
    rhs.addRuleRef(this) ;
    rhs.isConsequent() ;
    rb.ruleList.addElement(this) ; // se agrega a la lista de Reglas
    truth = null ;}
  public Rule(RuleBase Rb, String Name, Clause lhs1, Clause lhs2,
                           Clause lhs3, Clause lhs4,
                           Clause lhs5, Clause lhs6,
                           Clause lhs7, Clause lhs8,
                           Clause rhs) {
    rb = Rb ;
    name = Name ;
    antecedents = new Clause[8] ;
    antecedents[0] = lhs1 ;
    lhs1.addRuleRef(this) ;
    antecedents[1] = lhs2 ;
    lhs2.addRuleRef(this) ;
    antecedents[2] = lhs3 ;
    lhs3.addRuleRef(this) ;
    antecedents[3] = lhs4 ;
    lhs4.addRuleRef(this) ;
    antecedents[4] = lhs5 ;
    lhs5.addRuleRef(this) ;
    antecedents[5] = lhs6 ;
    lhs6.addRuleRef(this) ;
    antecedents[6] = lhs7 ;
    lhs7.addRuleRef(this) ;
    antecedents[7] = lhs8 ;
    lhs8.addRuleRef(this) ;
    consequent = rhs ;
    rhs.addRuleRef(this) ;
    rhs.isConsequent() ;
    rb.ruleList.addElement(this) ; // se agrega a la lista de Reglas
    truth = null ;}
  public Rule(RuleBase Rb, String Name, Clause lhs1, Clause lhs2,
                           Clause lhs3, Clause lhs4,
                           Clause lhs5, Clause lhs6,
                           Clause lhs7, Clause lhs8,
                           Clause lhs9,
                           Clause rhs) {
    rb = Rb ;
```

```
    name = Name ;
    antecedents = new Clause[9] ;
    antecedents[0] = lhs1 ;
    lhs1.addRuleRef(this) ;
    antecedents[1] = lhs2 ;
    lhs2.addRuleRef(this) ;
    antecedents[2] = lhs3 ;
    lhs3.addRuleRef(this) ;
    antecedents[3] = lhs4 ;
    lhs4.addRuleRef(this) ;
    antecedents[4] = lhs5 ;
    lhs5.addRuleRef(this) ;
    antecedents[5] = lhs6 ;
    lhs6.addRuleRef(this) ;
    antecedents[6] = lhs7 ;
    lhs7.addRuleRef(this) ;
    antecedents[7] = lhs8 ;
    lhs8.addRuleRef(this) ;
    antecedents[8] = lhs9 ;
    lhs9.addRuleRef(this) ;
    consèquent = rhs ;
    rhs.addRuleRef(this) ;
    rhs.isConsequent() ;
    rb.ruleList.addElement(this) ; // se agrega a la lista de Reglas
    truth = null ;}
  public Rule(RuleBase Rb, String Name, Clause lhs1, Clause lhs2,
                           Clause lhs3, Clause lhs4,
                           Clause lhs5, Clause lhs6,
                           Clause lhs7, Clause lhs8,
                           Clause lhs9, Clause lhs10,
                           Clause rhs) {
    rb = Rb ;
    name = Name ;
    antecedents = new Clause[10] ;
    antecedents[0] = lhs1 ;
    lhs1.addRuleRef(this) ;
    antecedents[1] = lhs2 ;
    lhs2.addRuleRef(this) ;
    antecedents[2] = lhs3 ;
    lhs3.addRuleRef(this) ;
    antecedents[3] = lhs4 ;
    lhs4.addRuleRef(this) ;
    antecedents[4] = lhs5 ;
    lhs5.addRuleRef(this) ;
    antecedents[5] = lhs6 ;
    lhs6.addRuleRef(this) ;
    antecedents[6] = lhs7 ;
    lhs7.addRuleRef(this) ;
    antecedents[7] = lhs8 ;
    lhs8.addRuleRef(this) ;
    antecedents[8] = lhs9 ;
    lhs9.addRuleRef(this) ;
    antecedents[9] = lhs10 ;
    lhs10.addRuleRef(this) ;
    consequent = rhs ;
    rhs.addRuleRef(this) ;
```

```
    rhs.isConsequent() ;
    rb.ruleList.addElement(this) ;  //  se agrega a la lista de Reglas
    truth = null ;}

  long numAntecedents() { return antecedents.length; }

  // Metodo CheckRules(), es usado por forward chaining !
  // un valor de una variable es encontrado, se revisan todas las clauses
  // que hacen referencia a dicha variable, y luego las a las cuales hacen
  // referencia dichas clauses

  public static void checkRules(Vector clauseRefs) {
   Enumeration enum = clauseRefs.elements();
  while(enum.hasMoreElements()) {
   Clause temp = (Clause)enum.nextElement();
   Enumeration enum2 = temp.ruleRefs.elements() ;
   while(enum2.hasMoreElements()) {
      ((Rule)enum2.nextElement()).check() ; // Prueba la Regla
   }
  }
}

  // Metodo check(), Usado por Forward Chaining
 Boolean check() {  // Si un antecedente es true y rule no ha sido disparada
  for (int i=0 ; i < antecedents.length ; i++ ) {
  if (antecedents[i].truth == null) return null ;
  if (antecedents[i].truth.booleanValue() == true) {
     continue ;
  } else {
     return truth = new Boolean(false) ; //no dispara esta Regla
  }
  } // fin de for
  return truth = new Boolean(true) ;  // puede disparar esta regla
 }

 // Metodo >fire(), usado por forward chaining !
 // Al dispara esta Regla -- Realiza la clause consecuente
 // Si una variable cambia, se actualizan todas las clauses donde
 // esta hace referencia y todas las reglas que contienen a
 // dichas clauses

 void fire() {
  RuleBase.appendText("\n Dispara Regla: " + name ) ;
  truth = new Boolean(true) ;
  fired = true ;
  // Asigna el valor de la variable y actualiza las clauses
  consequent.lhs.setValue(consequent.rhs) ;
  // Ahora vuelve a probar todas las reglas en las que las clauses cambiaron
  checkRules(consequent.lhs.clauseRefs) ;
 }

 // determina si una Regla es true o false
 // Por medio de recursividad al probar si cada una de las clauses
 // antecedentes clauses son true
 // Si alguna es false, la Regla es false
 Boolean backChain(TextArea TA)
```

```
{
  RuleBase.appendText("\nEvaluating rule " + name) ;
  for (int i=0 ; i < antecedents.length ; i++) { // prueba cada clause
   if (antecedents[i].truth == null) rb.backwardChain(antecedents[i].lhs.name,TA);
   if (antecedents[i].truth == null) { // No se puede probar si es true o false
   TA.append("\nValor x:******"+ antecedents[i].lhs.name +"***");
      String RespX= this.waitForAnswer(antecedents[i].lhs);
      antecedents[i].lhs.setValue(RespX);
      TA.append("\n Respuesta del dialogo:"+ RespX);
      truth = antecedents[i].check() ; // redundante?
   } // fin if
   if (antecedents[i].truth.booleanValue() == true) {
      continue ;   // prueba el antecedent siguiente
   } else {
      return truth = new Boolean(false) ; // Aborta, si alguna es false
   }
  } // fin for
  return truth = new Boolean(true) ; // todos los antecedentes son true
}

// Despliega la regla en formato de texto
public void display(TextArea textArea) {
  textArea.append("\n" + name +": Si ") ;
  for(int i=0 ; i < antecedents.length ; i++) {
    Clause nextClause = antecedents[i] ;
    textArea.append(nextClause.lhs.name +
                    nextClause.cond.asString() +
                    nextClause.rhs + " ") ;
    if ((i+1) < antecedents.length) textArea.append("\n    y  ") ;
  }
  textArea.append("\n    Ent- ") ;
     textArea.append(consequent.lhs.name +
                    consequent.cond.asString() +
                    consequent.rhs + "\n") ;
}
public void Disp(TextArea textArea) {
  textArea.append("\n<rule>;" + name +";") ;
  for(int i=0 ; i < antecedents.length ; i++) {
    Clause nextClause = antecedents[i] ;
    textArea.append(nextClause.lhs.name +"&"+
                    nextClause.cond.asString() +"&"+
                    nextClause.rhs ) ;
    if ((i+1) < antecedents.length) textArea.append(";") ;
  }
  textArea.append(";") ;
     textArea.append(consequent.lhs.name +"&"+
                    consequent.cond.asString() +"&"+
                    consequent.rhs) ;
}
 public void DispApp(TextArea textArea) {
 textArea.append("Rule " + name +" = new Rule(BaseReg, \""+name+"\",\n");
 textArea.append("                          new Clause(");
 for(int i=0 ; i < antecedents.length ; i++) {
   Clause nextClause = antecedents[i] ;
   textArea.append(nextClause.lhs.name +", "+
                    nextClause.cond.asStringX() +" ,\""+
```

```
                nextClause.rhs ) ;
    if ((i+1) < antecedents.length) textArea.append("\"),\n                    new Clause(") ;
   }
   textArea.append("\"),\n                    new Clause(") ;
    textArea.append(consequent.lhs.name +", "+
                consequent.cond.asStringX() +", \""+
                consequent.rhs + "\"));\n") ;
  textArea.append("BaseReg.RuleList.put("+name+".name,"+name+");\n\n");
 }

// Cuando una variable no es conocido su valor, se le
// pregunta al usuario informacion adicional acerca de dicha
// variable
 public String waitForAnswer(RuleVariable RVarZZ){
//************************************************************
  Frame frame= new Frame("nuevo");
  BOX2 = new Dialog1(frame);
  BOX2.setModal(true);
  BOX2.label1.setText(RVarZZ.promptText);
  BOX2.setLocation(250,250);
  Enumeration DNum = RVarZZ.labels.elements();
  while (DNum.hasMoreElements()) {
    BOX2.list1.addItem((String)DNum.nextElement());}
  BOX2.show();
  String Resp = BOX2.list1.getSelectedItem();
//************************************************************
  if (Resp == "") {Resp= null;}
  return Resp;}
};
```

/*Clase RuleBase, es la clase principal para la construccion de la base de conocimiento del sistema experto, se conforma de un conjunto de reglas, un conjunto de variables, y cadenas de visualización.
***/**

```
import java.util.*;
import java.io.*;
import java.awt.* ;
public class RuleBase {
public String name ;
public Hashtable variableList;   // Lista de variables en la Base de reglas
public Hashtable RuleList;
public Clause clauseVarList[];
public Vector ruleList ;        // lista de todas las reglas
public String MsgIntro;
public String MsgGen;
public String MsgFinal;
Vector conclusionVarList ;      // cola de variables
Rule rulePtr ;                  // Apuntador de Regla de trabajo
Clause clausePtr ;              // Apuntador de clause analizando

public Stack goalClauseStack;  // Pila de goalas y subgoals
public static TextArea TAR = new TextArea();
public void setDisplay(TextArea txtArea) {TAR = txtArea;}
public RuleBase(String Name) { name = Name; }
public static void appendText(String text) { TAR.appendText(text); }
```

```
 // Para propositos de monitoreo - despliega todas las variables y sus valores
 public void displayVariables(TextArea textArea) {
  Enumeration enum = variableList.elements() ;
  while(enum.hasMoreElements()) {
    RuleVariable temp = (RuleVariable)enum.nextElement() ;
    if (temp.value != null){textArea.appendText("\n" + temp.name + " valor = " + temp.value) ;}
   }
 }

// Para propositos de monitoreo - despliega todas las reglas en formato de texto
 public void displayRules(TextArea textArea) {
   textArea.appendText("\n" + name + " Base de Reglas: " + "\n");
   Enumeration enum = ruleList.elements() ;
   while(enum.hasMoreElements()) {
    Rule temp = (Rule)enum.nextElement() ;
    temp.display(textArea) ;}
}

// Para propositos de monitoreo - despliega todas las en conflicto
public void displayConflictSet(Vector ruleSet) {
  TAR.appendText("\n" + " -- Conjunto de reglas en conflicto:\n");
  Enumeration enum = ruleSet.elements() ;
  while(enum.hasMoreElements()) {
    Rule temp = (Rule)enum.nextElement() ;
    TAR.appendText(temp.name + "(" + temp.numAntecedents()+ "),\n " ) ;}
}

// Resetea la Base de reglas para otro ciclo de inferencia
// esto se hace poniendo los valores de todas las variables
// en null
public void reset() {
  Enumeration enum = variableList.elements() ;
  while(enum.hasMoreElements()) {
    RuleVariable temp = (RuleVariable)enum.nextElement() ;
    temp.setValue(null);}
    enum = ruleList.elements();
    while( enum.hasMoreElements()) {
    Rule RX = (Rule)enum.nextElement();
    RX.truth = null ;
    RX.fired = false ;}
}

 // Para todas las clauses Consequentes que hacen referencia a esta meta (goalVar)
 // SE intenta encontrar la meta por medio de las reglas que han sido disparadas
 //    Si la regla es true se extrae de la pila, y se le asigna el valor predeterminado
 //    Si la regla es false se extrae de la pila, y continua
 //    Si la regla es  null No se puede encontrar un valor para la meta
 //
public void backwardChain(String goalVarName,TextArea TA){
 RuleVariable goalVar = (RuleVariable)variableList.get(goalVarName);
 Enumeration goalClauses = goalVar.clauseRefs.elements() ;
 while (goalClauses.hasMoreElements()) {
    Clause goalClause = (Clause)goalClauses.nextElement() ;
    if (goalClause.consequent.booleanValue() == false) continue ;
    goalClauseStack.push(goalClause) ;
```

```
    Rule goalRule = goalClause.getRule();
    Boolean ruleTruth = goalRule.backChain(TA) ; // Encuentra regla con valor truth
    if (ruleTruth == null) {
     TA.appendText("\nRegla " + goalRule.name + " es nula, no se puede determinar su valor
truth");
    } else if (ruleTruth.booleanValue() == true) {
     // La Regla es OK, se le asigna el valor del consecuente a la variable
     goalVar.setValue(goalClause.rhs) ;
     goalVar.setRuleName(goalRule.name) ;
     goalClauseStack.pop() ; // Borra elemento de la pila de subgoal
     TA.appendText("\nRegla " + goalRule.name + " es true, Asignando " + goalVar.name + ": = " +
goalVar.value);
     if (goalClauseStack.empty() == true) {
      TA.appendText("\n\n +++ Se encontro Solucion para la Meta: " + goalVar.name +" es: "+
goalVar.value);
       break ; // Hasta ahora solo encuetra la primera solution, y se detiene
     }
    } else {
     goalClauseStack.pop() ; // Borra elemento de la pila subgoal
    TA.appendText("\nRegla " + goalRule.name + " es false, no se puede asignar " +
goalVar.name);
    }
  } // fin ciclo while;
 if (goalVar.value == null) {
    TA.appendText("\n\n +++ No se encontro Solucion para la Meta: " + goalVar.name +" ???");
  }
 }
// Metodo match(boolean test), usado por forward chaining
// determina cuales reglas pueden dispararse y regresa un Vector

 public Vector match(boolean test) {
   Vector matchList = new Vector() ;
   Enumeration enum = ruleList.elements() ;
   while (enum.hasMoreElements()) {
     Rule testRule = (Rule)enum.nextElement() ;
     if (test) testRule.check() ; // prueba los antecedentes de la regla
     if (testRule.truth == null) continue ;
     // dispara la regla una sola vez
     if ((testRule.truth.booleanValue() == true) &&
       (testRule.fired == false)) matchList.addElement(testRule) ;}
   displayConflictSet(matchList) ;
   return matchList ;}

// Metodo selectRule(Ruelset), es usado por forward chaining
// recibe el conjunto de reglas en conflicto y
// selecciona una regla para disparar
 public Rule selectRule(Vector ruleSet) {
   Enumeration enum = ruleSet.elements() ;
   long numClauses ;
   Rule nextRule ;
   Rule bestRule = (Rule)enum.nextElement() ;
   long max = bestRule.numAntecedents() ;
   while (enum.hasMoreElements()) {
     nextRule = (Rule)enum.nextElement() ;
     if ((numClauses = nextRule.numAntecedents()) > max) {
       max = numClauses ;
```

```
        bestRule = nextRule ;}
    }
    return bestRule ;
  }

  //*************************************************
  //*     Encadenamiento hacia Adelante             *
  //*************************************************
  public void forwardChain() {
    Vector conflictRuleSet = new Vector() ;
    //primero prueba todas las reglas, en base a los datos iniciales
    conflictRuleSet = match(true); // Ve que reglas pueden dispararse
    while(conflictRuleSet.size() > 0) {
      Rule selected = selectRule(conflictRuleSet); // selecciona la mejor regla
      selected.fire() ; // dispara la regla
                  // por medio de la accion de asiganmiento al consequent
                  // actualiza todas las clauses y reglas
      conflictRuleSet = match(false); //Ve que reglas pueden dispararse
      TAR.appendText("\n Inicia Ciclo de Inferencia con Encadenamiento hacia Adelante\n" );
      displayVariables(TAR) ; // despliega enlaces de variables
      TAR.appendText("\n Fin de Ciclo de Inferencia con Encadenamiento hacia Adelante\n" );
    }
  }
}
```

/*Clase Applicacion1, es la clase principal para la ejecución del sistema
*/

```
import com.sun.java.swing.UIManager;
import java.awt.*;
public class Application1 {
  boolean packFrame = false;
  //Constructor de la Aplicación
  public Application1() {
    Frame1 frame = new Frame1();
    //Valida los Frames a los tamaños iniciales
    if (packFrame)
      frame.pack();
    else
      frame.validate();
    //Centra la ventana
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height)
      frameSize.height = screenSize.height;
    if (frameSize.width > screenSize.width)
      frameSize.width = screenSize.width;
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
    frame.setVisible(true);}
//Método Principal
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(new com.sun.java.swing.plaf.windows.WindowsLookAndFeel());
      //UIManager.setLookAndFeel(new com.sun.java.swing.plaf.motif.MotifLookAndFeel());
```

```
    //UIManager.setLookAndFeel(new com.sun.java.swing.plaf.metal.MetalLookAndFeel());
   }
   catch (Exception e) {
   }
   new Application1();
  }
}

/* Clase DialError, es para despliegue de mensajes de error, esto se hace
en un dialogo independiente, según el tipo de error, un mensaje determinado
se despliega en el cuadro del dialogo
*/
import java.awt.*;
import java.awt.event.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;
public class DialError extends Dialog {
  Panel panel1 = new Panel();
  XYLayout xYLayout1 = new XYLayout();
  Button button1 = new Button();
  TextControl MsgErr = new TextControl();
public DialError(Frame frame, String title, boolean modal) {
   super(frame, title, modal);
   enableEvents(AWTEvent.WINDOW_EVENT_MASK);
   try {
    jbInit();
    add(panel1);
    pack();   }
   catch (Exception ex) {
    ex.printStackTrace();}
 }
 public DialError(Frame frame) {
  this(frame, "", false);  }
 public DialError(Frame frame, boolean modal) {
  this(frame, "", modal);  }
 public DialError(Frame frame, String title) {
  this(frame, title, false);}

 void jbInit() throws Exception {
  xYLayout1.setHeight(124);
  button1.setLabel("Aceptar");
  button1.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    button1_actionPerformed(e);}
  });
  MsgErr.setAlignment(borland.jbcl.util.Alignment.CENTER | borland.jbcl.util.Alignment.MIDDLE);
  MsgErr.setText("Mensaje de Error");
  panel1.setLocale(java.util.Locale.getDefault());
  xYLayout1.setWidth(319);
  panel1.setLayout(xYLayout1);
  panel1.add(button1, new XYConstraints(109, 81, 101, 28));
  panel1.add(MsgErr, new XYConstraints(15, 25, 288, 42));}

protected void processWindowEvent(WindowEvent e) {
 if (e.getID() == WindowEvent.WINDOW_CLOSING) {
```

```
    cancel();}
   super.processWindowEvent(e);}

  void cancel() {
   dispose();}

  void button1_actionPerformed(ActionEvent e) {
   dispose();}
}
```

/*Clase Frame1 esta es la clase principal del sistema, esta clase tiene todo lo necesario para el desarrollo de sistemas experto. Interectua con todas las demás, y es la que realiza la función principal del sistema.
***/**

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import borland.jbcl.control.*;
import borland.jbcl.layout.*;
import RuleVariable;
import RuleBase;
import Condition;
import Rule;
import Clause;
import java.io.*;
public class Frame1 extends DecoratedFrame {
  String valores= new String("");
  RuleBase RBAux =new RuleBase("RegAux");
  RuleBase RBAuxII= new RuleBase("LOAD");
  RuleBase RBNueva;
  RuleBase Transporte;
  RuleBase Otra;
  Hashtable BasesCon = new Hashtable();
  int ContClause = 1;
  Clause Clause1;
  Clause Clause2;
  Clause Clause3;
  Clause Clause4;
  Clause Clause5;
  Clause Clause6;
  Clause Clause7;
  Clause Clause8;
  Clause Clause9;
  Clause Clause10;
  Clause ClauseR;
  String MsgIntro = new String("Introduccion");
  String MsgFinal = new String("Msg Final");
// Dialogo de Error //
  Frame frame= new Frame();
  DialError MsgError = new DialError(frame,"                    E r r o r",true);
  Nombre NOMBOX;
  String currFileName = null;
  String currAppletName = null;
  boolean dirty = false;
```

```
boolean ValConsec = false;
String  CR = new String("\n");
//Construct the frame
BorderLayout borderLayout1 = new BorderLayout();
BevelPanel bevelPanel1 = new BevelPanel();
MenuBar menuBar1 = new MenuBar();
Menu menuFile = new Menu();
MenuItem menuFileExit = new MenuItem();
Menu menuHelp = new Menu();
MenuItem menuHelpAbout = new MenuItem();
StatusBar statusBar = new StatusBar();
TabsetPanel tabsetPanel1 = new TabsetPanel();
Panel VAR = new Panel();
Panel CLAUS = new Panel();
Panel ENC = new Panel();
Panel BC = new Panel();
Panel CONF =new Panel();
Label BRLabel = new Label();
XYLayout xYLayout1 = new XYLayout();
Label BCActual = new Label();
Label label6 = new Label();
Label label4 = new Label();
Label label5 = new Label();
TextField NomVartxt = new TextField();
TextField ProVartxt = new TextField();
TextField OpVartxt = new TextField();
Button VarOK = new Button();
Button OpVarAgregar = new Button();
Button OpVarBorrar = new Button();
List OpList = new List();
List VarList = new List();
XYLayout xYLayout2 = new XYLayout();
XYLayout xYLayout3 = new XYLayout();
XYLayout xYLayout4 = new XYLayout();
XYLayout xYLayout5 = new XYLayout();
Label label3 = new Label();
Label label1 = new Label();
Label label2 = new Label();
Label label7 = new Label();
Label label8 = new Label();
Choice SelVarAnt = new Choice();
Choice SelOperAnt = new Choice();
Choice SelValAnt = new Choice();
Label label12 = new Label();
TextField NomRegTxt = new TextField();
List list3 = new List();
Label label9 = new Label();
Choice SelVarCons = new Choice();
Choice SelOperCons = new Choice();
Choice SelValCons = new Choice();
List RegList = new List();
TextArea textArea1 = new TextArea();
Button CRBAND = new Button();
Button CRBEdClause = new Button();
Button CRBOK = new Button();
Label label10 = new Label();
```

```
Label label13 = new Label();
Label label14 = new Label();
Label label15 = new Label();
Choice ChSelBC = new Choice();
Choice ChSelVar = new Choice();
Choice ChSelVal = new Choice();
TextArea textArea3 = new TextArea();
Button button1 = new Button();
Button button2 = new Button();
Button button3 = new Button();
Choice ChSelMeta = new Choice();
Label label17 = new Label();
Label label16 = new Label();
Label label18 = new Label();
Choice BCSelBC = new Choice();
Label label19 = new Label();
Label label20 = new Label();
Label label21 = new Label();
List BCVarList = new List();
List BcListReg = new List();
TextArea BCText = new TextArea();
Button button4 = new Button();
Button CargaBC = new Button();
Button GuardaBC = new Button();
List BCListReg = new List();
Label label22 = new Label();
Filer filer1 = new Filer();
Button button7 = new Button();
List CFGValList = new List();
Button button8 = new Button();
Choice CFGVar = new Choice();
Label label23 = new Label();
Label label24 = new Label();
Label label25 = new Label();
List CGFAppList = new List(4);
Label label11 = new Label();
Choice SelBCApplet = new Choice();
Label label26 = new Label();
Button button9 = new Button();
Button button10 = new Button();
Choice choice1 = new Choice();
Button GETURL = new Button();
TextField TxtURL = new TextField();
TransparentImage TIMAGEN = new TransparentImage();
TransparentImage IMGN2 = new TransparentImage();
TextArea textArea2 = new TextArea();
Label label27 = new Label();
MenuItem menuItem1 = new MenuItem();
MenuItem menuItem2 = new MenuItem();
Button GenApplet = new Button();

public Frame1() {
 try {
  jbInit();
 }
 catch (Exception e) {
```

```
    e.printStackTrace();
   }
  }
//Component initialization

 private void jbInit() throws Exception {
  this.setLayout(borderLayout1);
  this.setSize(new Dimension(748, 596));
  this.setTitle("Frame Title");
  menuFile.setLabel("Archivo");
  menuFileExit.setLabel("Salir");
  menuFileExit.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
    fileExit_actionPerformed(e);
   }
  });
  menuHelp.setLabel("Ayuda");
  menuHelpAbout.setLabel("Acerca de WebXpert");
  VAR.setLayout(xYLayout1);
  BRLabel.setFont(new Font("Dialog", 0, 15));
  BRLabel.setText("Base Reglas:");
  BCActual.setText("BcActual");
  label6.setText("Para modificar una variable, seleccionela de la lista siguiente haciendo " +
  "doble clic");
  BC.setLayout(xYLayout4);
  ENC.setLayout(xYLayout3);
  CLAUS.setLayout(xYLayout2);
  CONF.setLayout(xYLayout5);
  VarOK.setActionCommand("OK");
  VarOK.setLabel("OK");
  OpVarAgregar.setActionCommand("Agrega");
  OpVarAgregar.setLabel("Agrega");
  OpVarBorrar.setActionCommand("Borrar");
  OpVarBorrar.setLabel("Borrar");
  label3.setText("Nombre:");
  label1.setText("Antecedentes");
  label2.setText("Variable");
  label7.setText("Operador");
  label8.setText("Valor");
  label12.setText("Nombre de la Regla");
  label9.setText("Consecuente");
  CRBAND.setLabel("- Y -");
  CRBAND.disable();
  CRBEdClause.setLabel("Editar Clause");
  CRBOK.setLabel("OK");
  CRBOK.disable();
  label10.setText("Reglas");
  label13.setText("Selec Base Reglas");
  label14.setText("Variable");
  label15.setText("Valor");
  textArea3.setFont(new Font("SansSerif", 0, 14));
  textArea3.setText("textArea2");
  button1.setLabel("Enc. Adel.");
  button2.setLabel("Enc. Atras.");
  button3.setLabel("Reset");
  label17.setText("Meta");
```

```
    label16.setText("Proceso");
    label18.setText("Nombre de la Base de Reglas");
    label19.setText("Variables");
    label20.setText("Valores");
    label21.setText("Lista de Reglas");
    BCText.setFont(new Font("Dialog", 0, 16));
    BCText.setText("textArea2");
    CargaBC.setLabel("Carga BC");
    GuardaBC.setLabel("Guarda BC");
    label22.setText("Base de Conocimiento");
    button4.setLabel("Nueva");
    label4.setText("Prompt:");
    label5.setText("Opciones:");
    menuHelpAbout.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        helpAbout_actionPerformed(e);
      }
    });
    bevelPanel1.setLayout(null);
    textArea1.setFont(new Font("SansSerif", 1, 16));
    textArea1.setForeground(Color.white);
    BCActual.setForeground(Color.blue);
    BCActual.setFont(new Font("SansSerif", 1, 20));
    TIMAGEN.setAlignment(borland.jbcl.util.Alignment.CENTER |
borland.jbcl.util.Alignment.MIDDLE);
    TIMAGEN.setBackground(new Color(192, 188, 192));
    IMGN2.setBackground(new Color(192, 188, 192));
    IMGN2.setAlignment(borland.jbcl.util.Alignment.CENTER | borland.jbcl.util.Alignment.MIDDLE);
    menuFile.add(menuItem1);
    menuFile.add(menuItem2);
    menuFile.add(menuFileExit);
    menuHelp.add(menuHelpAbout);
    menuBar1.add(menuFile);
    menuBar1.add(menuHelp);
    this.setMenuBar(menuBar1);
    this.add(statusBar, BorderLayout.SOUTH);
    this.add(bevelPanel1, BorderLayout.CENTER);
    bevelPanel1.add(tabsetPanel1, null);
    tabsetPanel1.add(VAR, "Variables");
    VAR.add(BRLabel, new XYConstraints(25, 19, 91, -1));
    VAR.add(BCActual, new XYConstraints(121, 12, 285, 32));
    VAR.add(label3, new XYConstraints(178, 92, 52, -1));
    VAR.add(label4, new XYConstraints(182, 142, 48, -1));
    VAR.add(label5, new XYConstraints(166, 191, 64, -1));
    VAR.add(label6, new XYConstraints(103, 463, 460, -1));
    VAR.add(NomVartxt, new XYConstraints(233, 89, 280, 26));
    VAR.add(ProVartxt, new XYConstraints(233, 139, 280, 26));
    VAR.add(OpVartxt, new XYConstraints(233, 189, 280, 25));
    VAR.add(OpList, new XYConstraints(233, 234, 280, 210));
    VAR.add(VarList, new XYConstraints(34, 90, 123, 356));
    VAR.add(VarOK, new XYConstraints(538, 89, 113, 31));
    VAR.add(OpVarAgregar, new XYConstraints(538, 189, 113, 31));
    VAR.add(OpVarBorrar, new XYConstraints(538, 139, 113, 31));
    VAR.add(label27, new XYConstraints(38, 68, 60, 21));

    tabsetPanel1.add(CLAUS, "CLAUSULAS/REGLAS");
```

```
CLAUS.add(SelVarAnt, new XYConstraints(32, 48, 147, 23));
CLAUS.add(SelOperAnt, new XYConstraints(207, 48, 77, 23));
CLAUS.add(SelValAnt, new XYConstraints(317, 48, 118, 23));
CLAUS.add(list3, new XYConstraints(32, 83, 407, 107));
CLAUS.add(SelVarCons, new XYConstraints(32, 228, 147, 24));
CLAUS.add(SelOperCons, new XYConstraints(207, 228, 77, 24));
CLAUS.add(SelValCons, new XYConstraints(317, 228, 118, 24));
CLAUS.add(RegList, new XYConstraints(32, 283, 149, 164));
CLAUS.add(textArea1, new XYConstraints(202, 283, 466, 164));
CLAUS.add(CRBAND, new XYConstraints(517, 128, 105, 31));
CLAUS.add(label1, new XYConstraints(198, 5, 80, 24));
CLAUS.add(label2, new XYConstraints(37, 26, 116, 23));
CLAUS.add(label7, new XYConstraints(199, 26, 71, 23));
CLAUS.add(label8, new XYConstraints(321, 26, 113, -1));
CLAUS.add(label9, new XYConstraints(198, 201, 80, 19));
CLAUS.add(NomRegTxt, new XYConstraints(488, 48, 173, 23));
CLAUS.add(CRBEdClause, new XYConstraints(517, 163, 105, 31));
CLAUS.add(CRBOK, new XYConstraints(517, 198, 105, 31));
CLAUS.add(label12, new XYConstraints(495, 26, 114, 23));
CLAUS.add(label10, new XYConstraints(198, 258, 76, 20));
tabsetPanel1.add(ENC, "Encadenamiento");
ENC.add(ChSelBC, new XYConstraints(20, 28, 149, 25));
ENC.add(ChSelVar, new XYConstraints(195, 28, 149, 25));
ENC.add(ChSelVal, new XYConstraints(370, 28, 149, 25));
ENC.add(textArea3, new XYConstraints(20, 92, 500, 388));
ENC.add(button1, new XYConstraints(581, 28, 99, 26));
ENC.add(button2, new XYConstraints(581, 63, 99, 26));
ENC.add(button3, new XYConstraints(581, 98, 99, 26));
ENC.add(ChSelMeta, new XYConstraints(528, 173, 174, 27));
ENC.add(label13, new XYConstraints(20, 6, 147, 22));
ENC.add(label14, new XYConstraints(195, 6, 146, 22));
ENC.add(label15, new XYConstraints(370, 5, 39, 22));
ENC.add(label17, new XYConstraints(597, 153, 38, 24));
ENC.add(label16, new XYConstraints(22, 71, 101, 20));
tabsetPanel1.add(BC, "Base de Conocimiento");
BC.add(BCSelBC, new XYConstraints(15, 30, 168, 24));
BC.add(button4, new XYConstraints(568, 84, 111, 34));
BC.add(CargaBC, new XYConstraints(568, 129, 111, 34));
BC.add(GuardaBC, new XYConstraints(568, 174, 111, 34));
BC.add(BCVarList, new XYConstraints(15, 78, 345, 139));
BC.add(BCListReg, new XYConstraints(385, 78, 162, 139));
BC.add(BCText, new XYConstraints(15, 254, 534, 181));
BC.add(label18, new XYConstraints(15, 4, 149, 20));
BC.add(label19, new XYConstraints(15, 55, 71, 20));
BC.add(label20, new XYConstraints(200, 55, 63, 20));
BC.add(label21, new XYConstraints(385, 55, 92, 20));
BC.add(label22, new XYConstraints(23, 229, 129, 23));
BC.add(button9, new XYConstraints(568, 254, 111, 34));
BC.add(button10, new XYConstraints(568, 300, 111, 34));
BC.add(choice1, new XYConstraints(20, 445, 461, 23));
tabsetPanel1.add(CONF, "Opciones");
CONF.add(button7, new XYConstraints(548, 63, 104, 29));
CONF.add(CFGValList, new XYConstraints(39, 67, 180, 152));
CONF.add(button8, new XYConstraints(548, 104, 104, 29));
CONF.add(CFGVar, new XYConstraints(39, 37, 180, 24));
CONF.add(label23, new XYConstraints(40, 14, 311, 20));
```

```
    CONF.add(label24, new XYConstraints(45, 286, 229, 26));
    CONF.add(label25, new XYConstraints(369, 15, 349, 16));
    CONF.add(CGFAppList, new XYConstraints(39, 320, 109, 99));
    CONF.add(label11, new XYConstraints(284, 258, 156, 30));
    CONF.add(SelBCApplet, new XYConstraints(516, 325, 172, 27));
    CONF.add(label26, new XYConstraints(516, 297, 91, 26));
    CONF.add(GETURL, new XYConstraints(548, 146, 104, 29));
    CONF.add(TxtURL, new XYConstraints(509, 206, 210, 28));
    CONF.add(textArea2, new XYConstraints(425, 426, 294, 70));
    CONF.add(IMGN2, new XYConstraints(169, 320, 251, 180));
    CONF.add(TIMAGEN, new XYConstraints(237, 46, 264, 186));
    CONF.add(GenApplet, new XYConstraints(548, 373, 104, 29));
    BC.add(BCListReg, new XYConstraints(390, 77, 148, 137));
    CGFAppList.addItem("Forma5");
    CGFAppList.addItem("Forma6");
    CGFAppList.addItem("Forma7");
    CGFAppList.addItem("Forma8");
    GuardaBC.enable(false);
    SelVarAnt.removeAll();
    SelOperCons.removeAll();
    ChSelVar.removeAll();
    ChSelMeta.removeAll();
    BCSelBC.removeAll();
    ChSelVal.removeAll();
    SelOperAnt.addItem("=");
    SelOperAnt.addItem("<");
    SelOperAnt.addItem(">");
    SelOperAnt.addItem("!=");
    SelOperCons.addItem("=");
    SelOperCons.addItem("<");
    SelOperCons.addItem(">");
    SelOperCons.addItem("!=");

   filer1.setFrame(this);

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
// Definiones necesarias para la base de conocimiento
//Inicializacion de la base de conocimiento Transportes

   RBAux.variableList = new Hashtable();
   RBAuxII.variableList = new Hashtable();
ChSelBC.addItem(Transporte.name);
  BCSelBC.addItem(Transporte.name);
GenApplet.setLabel("Genera Applet");
    GenApplet.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        GenApplet_actionPerformed(e);
      }
    });
    menuItem2.setLabel("Guarda BC");
    menuItem1.setLabel("Abrir BC");
    RBAux.reset();
    this.setTitle(RBAux.name);
    Otra.RuleList.put(REG6.name,REG6);
SelBCApplet.addItem(Otra.name);
```

```
  button4.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      button4_actionPerformed(e);
    }
  });
  label27.setText("Variables:");
  CGFAppList.setFont(new Font("Dialog", 0, 16));
  list3.setFont(new Font("SansSerif", 0, 15));
  RegList.setFont(new Font("Dialog", 0, 14));
  VarList.setFont(new Font("SansSerif", 0, 14));
  IMGN2.setImageName("C:\\jbuilder2\\myprojects\\WX4\\Imagenes\\car3.jpg");
  CGFAppList.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      CGFAppList_actionPerformed(e);
    }
  });
  textArea2.setText("textArea2");
  OpVartxt.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyTyped(KeyEvent e) {
      OpVartxt_keyTyped(e);
    }
  });
  NomRegTxt.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      NomRegTxt_actionPerformed(e);
    }
  });
  TIMAGEN.setImageName("WX4\\Imagenes\\car5.jpg");
  TxtURL.setText("txtUrl");
  GETURL.setLabel("URL");
  GETURL.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      GETURL_actionPerformed(e);
    }
  });
  CFGVar.addItemListener(new java.awt.event.ItemListener() {
    public void itemStateChanged(ItemEvent e) {
      CFGVar_itemStateChanged(e);
    }
  });
  button10.setLabel("Msg Final");
  button10.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      button10_actionPerformed(e);
    }
  });
  button9.setLabel("Msg Intro");
  button9.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      button9_actionPerformed(e);
    }
  });
  label26.setText("Base de Reglas");
  SelBCApplet.addItemListener(new java.awt.event.ItemListener() {
    public void itemStateChanged(ItemEvent e) {
```

```
    SelBCApplet_itemStateChanged(e);
  }
 });
 label11.setForeground(Color.blue);
 label11.setText("Genera Applet");
 label11.setFont(new Font("SansSerif", 1, 20));
 label25.setText("Oprima OK para Aceptar, Seleccion para escoger otra Imagen");
 label24.setText("Seleccion del Applet para correr en WEB");
 label23.setText("Asignacion de Imagenes a atributos de las  variables");
 button8.setLabel("OK");
 button7.setLabel("Imagen ?");
 textArea1.setBackground(new Color(0, 100, 255));
 tabsetPanel1.setBounds(new Rectangle(1, 0, 738, 546));
 bevelPanel1.setBackground(Color.darkGray);
 NomRegTxt.addKeyListener(new java.awt.event.KeyAdapter() {
  public void keyTyped(KeyEvent e) {
    NomRegTxt_keyTyped(e);
  }
 });
 GuardaBC.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    GuardaBC_actionPerformed(e);
  }
 });
 CargaBC.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    CargaBC_actionPerformed(e);
  }
 });
 button3.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    button3_actionPerformed(e);
  }
 });
 button2.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    button2_actionPerformed(e);
  }
 });
 button1.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    button1_actionPerformed(e);
  }
 });
 ChSelMeta.addItemListener(new java.awt.event.ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    ChSelMeta_itemStateChanged(e);
  }
 });
 ChSelVal.addItemListener(new java.awt.event.ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    ChSelVal_itemStateChanged(e);
  }
 });
 ChSelVar.addItemListener(new java.awt.event.ItemListener() {
  public void itemStateChanged(ItemEvent e) {
```

```
    ChSelVar_itemStateChanged(e);
   }
  });
  ChSelBC.addItemListener(new java.awt.event.ItemListener() {
   public void itemStateChanged(ItemEvent e) {
    ChSelBC_itemStateChanged(e);
   }
  });
  RegList.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    RegList_actionPerformed(e);
   }
  });
  SelVarCons.addItemListener(new java.awt.event.ItemListener() {
   public void itemStateChanged(ItemEvent e) {
    SelVarCons_itemStateChanged(e);
   }
  });
  SelValAnt.addItemListener(new java.awt.event.ItemListener() {
   public void itemStateChanged(ItemEvent e) {
    SelValAnt_itemStateChanged(e);
   }
  });
  SelVarAnt.addItemListener(new java.awt.event.ItemListener() {
   public void itemStateChanged(ItemEvent e) {
    SelVarAnt_itemStateChanged(e);
   }
  });
  BCSelBC.addItemListener(new java.awt.event.ItemListener() {
   public void itemStateChanged(ItemEvent e) {
    BCSelBC_itemStateChanged(e);
   }
  });
  CRBOK.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    CRBOK_actionPerformed(e);
   }
  });
  CRBEdClause.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    CRBEdClause_actionPerformed(e);
   }
  });
  CRBAND.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    CRBAND_actionPerformed(e);
   }
  });
  OpVarBorrar.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    OpVarBorrar_actionPerformed(e);
   }
  });
  VarList.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    VarList_actionPerformed(e);
```

```
    }
   });
   OpList.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
     OpList_actionPerformed(e);
    }
   });
   OpVarAgregar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
     OpVarAgregar_actionPerformed(e);
    }
   });
   VarOK.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
     VarOK_actionPerformed(e);
    }
   });
/*  RBAux.reset();
  this.setTitle(Otra.name);
  Otra.RuleList.put(REG6.name,REG6);
   SelBCApplet.addItem(Transporte.name);
   SelBCApplet.addItem(Otra.name); */

  }
  //File | Exit action performed
  public void fileExit_actionPerformed(ActionEvent e) {
   System.exit(0);}
//Help | About action performed

  public void helpAbout_actionPerformed(ActionEvent e) {
   Frame1_AboutBox dlg = new Frame1_AboutBox(this);
   Dimension dlgSize = dlg.getPreferredSize();
   Dimension frmSize = getSize();
   Point loc = getLocation();
   dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height - dlgSize.height) / 2 +
loc.y);
   dlg.setModal(true);
   dlg.show();
  }

  void BCSelBC_itemStateChanged(ItemEvent e) {
   button9.setLabel("Msg Intro");
   GuardaBC.enable(true);
   BCText.setText("");
   BCVarList.removeAll();
  // BCListVal.removeAll();
   BCListReg.removeAll();
   String BCN = BCSelBC.getSelectedItem();
   RBAux = (RuleBase)BasesCon.get(BCN);
   //BCText.append("<nombre>;"+RBAux.name);
   RuleVariable RV;
   Enumeration Vars = RBAux.variableList.elements();
   String VarNomAux;
   String LabelAux;
   while (Vars.hasMoreElements()) {
      BCText.append("\n");
```

```
      RV = (RuleVariable)Vars.nextElement();
      BCVarList.addItem(RV.name +"   :"+RV.getLabels());
      BCText.append("<variable>;"+RV.name);
      BCText.append(";"+RV.promptText +";");
      Enumeration Vals = RV.labels.elements();
      BCText.append(RV.getLabels());
       }

  Vars = RBAux.ruleList.elements();
  while (Vars.hasMoreElements()) {
     BCListReg.addItem(((Rule)Vars.nextElement()).name);
     }
 Enumeration enum = RBAux.ruleList.elements() ;
 while(enum.hasMoreElements()) {
  Rule temp = (Rule)enum.nextElement() ;
  temp.Disp(BCText) ;
 }

BCText.append("\n<msgin>;"+ RBAux.MsgIntro);
BCText.append("\n<msgg>;" + RBAux.MsgGen);
BCText.append("\n<msgf>;" + RBAux.MsgFinal);
BCText.append("\n<BCFIN>;"+ ".....");
currFileName = null;
}

void CRBAND_actionPerformed(ActionEvent e) {
 CRBAND.enable(false);
 RBAux= (RuleBase)BasesCon.get(this.getTitle());
 RuleVariable RCVar= (RuleVariable)RBAux.variableList.get(SelVarAnt.getSelectedItem());
 String Cond1= SelOperAnt.getSelectedItem();
 Condition RCCond = new Condition(Cond1);
  String Valor = SelValAnt.getSelectedItem();
  textArea1.append("\nCont Clause="+ContClause);
switch (ContClause){

   case 1:
   Clause1 = new Clause(RCVar,RCCond,Valor);
   ContClause = ContClause+1;
   list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
   break;

   case 2:
   Clause2 = new Clause(RCVar,RCCond,Valor);
   ContClause = ContClause+1;
   list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
   break;
   case 3:
   Clause3 = new Clause(RCVar,RCCond,Valor);
   ContClause = ContClause+1;
   list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
   break;
   case 4:
   Clause4 = new Clause(RCVar,RCCond,Valor);
   ContClause = ContClause+1;
   list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
   break;
```

```
        case 5:
        Clause5 = new Clause(RCVar,RCCond,Valor);
        ContClause = ContClause+1;
        list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
        break;
        case 6:
        Clause6 = new Clause(RCVar,RCCond,Valor);
        ContClause = ContClause+1;
        list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
        break;
        case 7:
        Clause7 = new Clause(RCVar,RCCond,Valor);
        ContClause = ContClause+1;
        list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
        break;
        case 8:
        Clause8 = new Clause(RCVar,RCCond,Valor);
        ContClause = ContClause+1;
        list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
        break;
        case 9:
        Clause9 = new Clause(RCVar,RCCond,Valor);
        ContClause = ContClause+1;
        list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
        break;
        case 10:
        Clause10 = new Clause(RCVar,RCCond,Valor);
        ContClause = ContClause+1;
        list3.addItem(SelVarAnt.getSelectedItem() + " " + Cond1 + " " + Valor);
        CRBAND.enable(false);
        break;
        case 11:
        CRBAND.enable(false);
        break;
        }
}

void button1_actionPerformed(ActionEvent e) {
  RBAux= (RuleBase)BasesCon.get(this.getTitle());
  RBAux.setDisplay(textArea3);
  textArea3.append("\n  Inicia ciclo de inferencia  \n Base de Reglas:" + RBAux.name);
  RBAux.displayVariables(textArea3);
  RBAux.forwardChain();
  textArea3.append("\n Fin de Ciclo de Inferencia  \n");
  RBAux.displayVariables(textArea3);
}

void button2_actionPerformed(ActionEvent e) {
  String Meta = ChSelMeta.getSelectedItem();
  RBAux= (RuleBase)BasesCon.get(this.getTitle());
  textArea3.append("\n  Inicia ciclo de inferencia.....Para Meta="+ Meta);

  RBAux.displayVariables(textArea3);
  RBAux.backwardChain(Meta, textArea3);
  textArea3.append("\n Fin de Ciclo de Inferencia  \n");
  RBAux.displayVariables(textArea3);
```

```
}

void button3_actionPerformed(ActionEvent e) {
  RBAux = (RuleBase)BasesCon.get(this.getTitle());
  textArea3.setText("");
  RBAux.reset();
  RBAux.displayVariables(textArea3);
}

void SelVarAnt_itemStateChanged(ItemEvent e) {
  CRBAND.enable();
  RBAux = (RuleBase)BasesCon.get(this.getTitle());
  String VarName1 = SelVarAnt.getSelectedItem();
  SelValAnt.removeAll();
  RuleVariable Rvar1 = (RuleVariable)RBAux.variableList.get(VarName1);
  Enumeration num1 = Rvar1.labels.elements();
  while(num1.hasMoreElements()){
    SelValAnt.addItem(((String)num1.nextElement()));
    }
}

void VarOK_actionPerformed(ActionEvent e) {
 if ((NomVartxt.getText()).equals("")) {
            MsgError.MsgErr.setText("Debe llenar el campo de Nombre de la Variable");
            MsgError.show();
            NomVartxt.requestFocus();}
 else{
     if((ProVartxt.getText()).equals("")){
            MsgError.MsgErr.setText("Debe llenar el campo de Prompt para Variable");
            MsgError.show();
            ProVartxt.requestFocus();}
     else{
         if ((OpList.getItemCount()) == 0){
            MsgError.MsgErr.setText("Debe llenar las opciones dela Variable");
            MsgError.show();
            OpVartxt.requestFocus();}
         else{


if (VarOK.getLabel()== "OK") {
   String Alfa = NomVartxt.getText();
   boolean Existe = false;
   for (int i= 0; i<VarList.getItemCount();i++){
      if (Alfa.equals(VarList.getItem(i))){Existe=true;
         MsgError.MsgErr.setText("El nombre de la variable \""+Alfa+"\" ya existe");
         MsgError.show();
         NomVartxt.requestFocus();}// if
           }//for
   if (Existe==false){

RBAux= (RuleBase)BasesCon.get(this.getTitle());
RuleVariable aux = new RuleVariable(NomVartxt.getText());
VarList.addItem(aux.name);
aux.setLabels(valores);
aux.setPromptText(ProVartxt.getText());
RBAux.variableList.put(aux.name, aux);
```

```
    valores = "";
    OpList.removeAll();
    NomVartxt.setText("");
    ProVartxt.setText("");
    SelVarAnt.addItem(aux.name);
    SelVarCons.addItem(aux.name);
    ChSelVar.addItem(aux.name);
    NomVartxt.requestFocus();
        } // if de Existe
    }

    if (VarOK.getLabel() == "Actualizar") {
    RBAux= (RuleBase)BasesCon.get(this.getTitle());
    String VV = NomVartxt.getText();
    RuleVariable aux = (RuleVariable)RBAux.variableList.get(VV);
    aux.setLabels(valores);
    aux.setPromptText(ProVartxt.getText());
    valores = "";
    OpList.removeAll();
    NomVartxt.setText("");
    ProVartxt.setText("");
    OpVartxt.setText("");
    VarOK.setLabel("OK");
    NomVartxt.requestFocus();
    }
    }
    }
    }
}

void OpVarAgregar_actionPerformed(ActionEvent e) {
    if ((OpVartxt.getText()).equals("")){
    MsgError.MsgErr.setText("Debe llenar el campo de Opciones");
    MsgError.show();
    OpVartxt.requestFocus();
    }
 else{
    String Alfa = OpVartxt.getText();
    boolean Existe = false;
    for (int i= 0; i<OpList.getItemCount();i++){
        if (Alfa.equals(OpList.getItem(i))){Existe=true;
                                OpVartxt.setText("");
                                OpVartxt.requestFocus();}}// cierre de if y for
    if (Existe==false) {
    OpList.addItem(OpVartxt.getText());
    if (valores == ""){ valores = OpVartxt.getText();}
    else {valores = valores+ "&" + OpVartxt.getText();}
    OpVartxt.setText("");
    //OpVartxt.getCursor();
    OpVartxt.requestFocus();
    }
  }
}
void OpVarBorrar_actionPerformed(ActionEvent e) {
 NomVartxt.setText("");
 ProVartxt.setText("");
```

```
  OpList.removeAll();
  OpVartxt.setText("");
  VarOK.setLabel("OK");
  valores="";
  NomVartxt.requestFocus();
 }

 void CRBOK_actionPerformed(ActionEvent e) {
  String Alfa = NomRegTxt.getText();
 boolean Existe = false;
    for (int i= 0; i<RegList.getItemCount();i++){
         if (Alfa.equals(RegList.getItem(i))){Existe=true;
             MsgError.MsgErr.setText("El nombre de la Regla \""+Alfa+"\" ya existe");
             MsgError.show();
             NomRegTxt.requestFocus();}// if
              }//for
   if (ContClause==1){
             MsgError.MsgErr.setText("No ha definido las premisas");
             MsgError.show();
             NomRegTxt.requestFocus();
             Existe= true;}// if

   if (SelValCons.getItemCount()== 0){
             MsgError.MsgErr.setText("No ha definido la consecuente");
             MsgError.show();
             SelVarCons.requestFocus();
             Existe= true;}// if

   if (Existe==false){
  RBAux= (RuleBase)BasesCon.get(this.getTitle());
  RuleVariable RCSVar= (RuleVariable)RBAux.variableList.get(SelVarCons.getSelectedItem());
  String Cond2= SelOperCons.getSelectedItem();
  Condition RCSCond = new Condition(Cond2);
  String ValorR = SelValCons.getSelectedItem();
 //ContClause= ContClause-1;    //*******************
  ClauseR = new Clause(RCSVar,RCSCond,ValorR);
  Rule Aux;
  //String NomReg = NomRegTxt.getText();
 if((NomRegTxt.getText()).equals("")){textArea1.setText("Error: debe poner un \n nombre de la
Regla");
                        NomRegTxt.requestFocus();}
  else{
     textArea1.setText("");
     ContClause=ContClause-1;
     textArea1.append("\nContClause="+ContClause+"\n"+"Nueva regla:"+ NomRegTxt.getText());
     CRBAND.disable();
     CRBOK.disable();
  switch (ContClause){

     case 1:
     Aux= new Rule(RBAux,NomRegTxt.getText(),Clause1,ClauseR);
     ContClause = 1;
     RegList.addItem(Aux.name);
     RBAux.RuleList.put(Aux.name, Aux);
     list3.removeAll();
     NomRegTxt.setText("");
```

```
break;

case 2:
Aux= new Rule(RBAux,NomRegTxt.getText(),Clause1,Clause2,ClauseR);
ContClause = 1;
RegList.addItem(Aux.name);
RBAux.RuleList.put(Aux.name, Aux);
list3.removeAll();
NomRegTxt.setText("");
break;

case 3:
Aux= new Rule(RBAux,NomRegTxt.getText(),Clause1,Clause2,Clause3,ClauseR);
ContClause = 1;
RegList.addItem(Aux.name);
RBAux.RuleList.put(Aux.name, Aux);
list3.removeAll();
NomRegTxt.setText("");
break;

case 4:
Aux= new Rule(RBAux,NomRegTxt.getText(),Clause1,Clause2,Clause3,Clause4,ClauseR);
ContClause = 1;
RegList.addItem(Aux.name);
RBAux.RuleList.put(Aux.name, Aux);
list3.removeAll();
NomRegTxt.setText("");
break;

case 5:
Aux= new Rule (RBAux,NomRegTxt.getText(),Clause1,Clause2, Clause3,Clause4,Clause5,
                ClauseR);
ContClause = 1;
RegList.addItem(Aux.name);
RBAux.RuleList.put(Aux.name, Aux);
list3.removeAll();
NomRegTxt.setText("");
break;

case 6:
Aux= new Rule(RBAux,NomRegTxt.getText(),Clause1,Clause2,Clause3,Clause4,Clause5,
                Clause6, ClauseR);
ContClause = 1;
RegList.addItem(Aux.name);
RBAux.RuleList.put(Aux.name, Aux);
list3.removeAll();
NomRegTxt.setText("");
break;

case 7:
Aux= new Rule(RBAux,NomRegTxt.getText(),Clause1,Clause2,Clause3,Clause4,Clause5,
                Clause6,Clause7,ClauseR);
ContClause = 1;
RegList.addItem(Aux.name);
RBAux.RuleList.put(Aux.name, Aux);
list3.removeAll();
```

```
      NomRegTxt.setText("");
      break;

      case 8:
      Aux= new Rule(RBAux,NomRegTxt.getText(),Clause1,Clause2,Clause3,Clause4,Clause5,
                    Clause6,Clause7,Clause8,ClauseR);
      ContClause = 1;
      RegList.addItem(Aux.name);
      RBAux.RuleList.put(Aux.name, Aux);
      list3.removeAll();
      NomRegTxt.setText("");
      break;

      case 9:
      Aux= new Rule(RBAux,NomRegTxt.getText(),Clause1,Clause2,Clause3,Clause4,Clause5,
                    Clause6,Clause7,Clause8,Clause9,ClauseR);
      ContClause = 1;
      RegList.addItem(Aux.name);
      RBAux.RuleList.put(Aux.name, Aux);
      list3.removeAll();
      NomRegTxt.setText("");
      break;

      case 10:
      Aux= new Rule(RBAux,NomRegTxt.getText(),Clause1,Clause2,Clause3,Clause4,Clause5,
                    Clause6,Clause7,Clause8,Clause9,Clause10,ClauseR);
      ContClause = 1;
      RegList.addItem(Aux.name);
      RBAux.RuleList.put(Aux.name, Aux);
      list3.removeAll();
      NomRegTxt.setText("");
      break;

    }
    boolean MetaEx = false;
    String Metax= ClauseR.lhs.name;
    for (int i= 0; i<ChSelMeta.getItemCount();i++){
      if (Metax.equals(ChSelMeta.getItem(i))){MetaEx=true;}
        }
      if (MetaEx==false){ChSelMeta.addItem(ClauseR.lhs.name);}

    }
  } //if de Existe nombre de Regla
  ValConsec= false;
}

void VarList_actionPerformed(ActionEvent e) {
  VarOK.setLabel("Actualizar");
  valores= "";
  String Vaux = VarList.getSelectedItem();
  RBAux= (RuleBase)BasesCon.get(this.getTitle());
  RuleVariable RVX = (RuleVariable)RBAux.variableList.get(Vaux);
  NomVartxt.setText(RVX.name);
  ProVartxt.setText(RVX.promptText);
  OpVartxt.setText("");
  OpList.removeAll();
```

```
      Enumeration Vals = RVX.labels.elements();
       while (Vals.hasMoreElements()){
       Vaux= (String)Vals.nextElement();
       if (valores == ""){ valores = Vaux;}
       else {valores = valores+ "&" + Vaux;}
         OpList.addItem(Vaux);
         }
   }

void ChSelBC_itemStateChanged(ItemEvent e) {
  // Actualiza valores de la BC en la interface grafica
  ContClause=1;
  list3.removeAll();
  OpList.removeAll();
  CFGVar.removeAll();
  CFGValList.removeAll();
  NomVartxt.setText("");
  ProVartxt.setText("");
  OpVartxt.setText("");
  textArea3.setText("");
  VarList.removeAll();
  RegList.removeAll();
  SelVarAnt.removeAll();
  SelValAnt.removeAll();
  SelVarCons.removeAll();
  SelValCons.removeAll();
  textArea1.setText("");
  NomRegTxt.setText("");
  String BCN = ChSelBC.getSelectedItem();
  this.setTitle(BCN);
  BCActual.setText(BCN);
  RuleBase RBAux = (RuleBase)BasesCon.get(BCN);
  RBAux.reset();
  ChSelVar.removeAll();
  ChSelVal.removeAll();
  ChSelMeta.removeAll();
  Enumeration Vars = RBAux.variableList.elements();
  String VarNomAux;
  while (Vars.hasMoreElements()) {
     VarNomAux = ((RuleVariable)Vars.nextElement()).name;
     ChSelVar.addItem(VarNomAux);
     SelVarAnt.addItem(VarNomAux);
     SelVarCons.addItem(VarNomAux);
    // ChSelMeta.addItem(VarNomAux);
     VarList.addItem(VarNomAux);
     CFGVar.addItem(VarNomAux);
     }
  Vars = RBAux.ruleList.elements();
  Rule Raux;
  while (Vars.hasMoreElements()) {
     Raux = (Rule)Vars.nextElement();
     boolean MetaEx = false;
     String Metax= Raux.consequent.lhs.name;
     for (int i= 0; i<ChSelMeta.getItemCount();i++){
       if (Metax.equals(ChSelMeta.getItem(i))){MetaEx=true;}
         }
```

```
            if (MetaEx==false){ChSelMeta.addItem(Raux.consequent.lhs.name);}
          RegList.addItem(Raux.name);
          }
        RBAux.displayVariables(textArea3);
// Actualiza valores de la BC en la interface grafica
    }

    void ChSelVar_itemStateChanged(ItemEvent e) {
      RBAux= (RuleBase)BasesCon.get(this.getTitle());
      String NVars = ChSelVar.getSelectedItem();
      ChSelVal.removeAll();
      RuleVariable VarReg = (RuleVariable)RBAux.variableList.get(NVars);
      Enumeration Valss = VarReg.labels.elements();
      while (Valss.hasMoreElements()) {
          ChSelVal.addItem(((String)Valss.nextElement()));
          }
    }

    void ChSelVal_itemStateChanged(ItemEvent e) {
      RBAux= (RuleBase)BasesCon.get(this.getTitle());
      String VarNom = ChSelVar.getSelectedItem();
      String VarVal = ChSelVal.getSelectedItem();
      RuleVariable Rvarx = (RuleVariable)RBAux.variableList.get(VarNom);
      Rvarx.setValue(VarVal);
      textArea3.append("\n" + Rvarx.name + " = " + VarVal);
      }

    void SelVarCons_itemStateChanged(ItemEvent e) {
      ValConsec= true;
      RBAux= (RuleBase)BasesCon.get(this.getTitle());
      String VarN = SelVarCons.getSelectedItem();
      SelValCons.removeAll();
      RuleVariable Rvar = (RuleVariable)RBAux.variableList.get(VarN);
      Enumeration Nums = Rvar.labels.elements();
      while(Nums.hasMoreElements()){
        SelValCons.addItem(((String)Nums.nextElement()));
        }
    }

    void RegList_actionPerformed(ActionEvent e) {
      RBAux= (RuleBase)BasesCon.get(this.getTitle());
      textArea1.setText("");
      Rule RegAux = (Rule)RBAux.RuleList.get(RegList.getSelectedItem());
      RegAux.display(textArea1);
      //Desplegar Clauses en List3 para poder editar y modificarlas
      for (int i=0 ; i < RegAux.antecedents.length ; i++ ) {
          list3.addItem(RegAux.antecedents[i].lhs.name+"    "+    RegAux.antecedents[i].cond    +    "
"+RegAux.antecedents[i].rhs);
      }
      }

    void SelBCApplet_itemStateChanged(ItemEvent e) {
      RuleBase AppRB = (RuleBase)BasesCon.get(SelBCApplet.getSelectedItem());
      textArea2.setText("");
      textArea2.append("<nombre>;"+AppRB.name);
      RuleVariable RVApp;
```

```
    Enumeration VarsApp = AppRB.variableList.elements();
    String VarAppAux;
    String LabelAppAux;
    while (VarsApp.hasMoreElements()) {
        textArea2.append("\n");
        RVApp = (RuleVariable)VarsApp.nextElement();
        textArea2.append("<variable>;"+RVApp.name);
        textArea2.append(";"+RVApp.promptText +";");
       // Enumeration ValsApp = RVApp.labels.elements();
        textArea2.append(RVApp.getLabels());
         }

 //  VarsApp = AppRB.ruleList.elements();
  // while (VarsApp.hasMoreElements()) {
      //BCListReg.addItem(((Rule)Vars.nextElement()).name);
 //     }
  Enumeration enumApp = AppRB.ruleList.elements() ;
  while(enumApp.hasMoreElements()) {
   Rule temp = (Rule)enumApp.nextElement() ;
   temp.Disp(textArea2) ;
  }

 textArea2.append("\n<msgin>;"+ RBAux.MsgIntro);
 textArea2.append("\n<msgg>;" + RBAux.MsgGen);
 textArea2.append("\n<msgf>;" + RBAux.MsgFinal);
 textArea2.append("\n<BCFIN>;"+ ".....");
 currAppletName = null;
 }
/* ----------------------------------------------------------
Metodo que guarda base de conocimiento en formato texto
----------------------------------------------------------*/
  boolean saveFile()
  {
   // Handle the case where we don't have a file name yet.
   if (currFileName == null) {
    return saveAsFile();
   }
   try
   {
    // Open a file of the current name.
    File file = new File (currFileName);
    // Create an output writer that will write to that file.
    // FileWriter handles international characters encoding conversions.
    FileWriter out = new FileWriter(file);
    String text ="<nombre>;"+file.getName()+BCText.getText();
    out.write(text);
    out.close();
    this.dirty = false;
    //updateCaption();
    return true;
   }
   catch (IOException e) {
    //statusBar.setText("Error saving "+currFileName);
   }
   return false;
  }
```

```
// fin metodo salvar archivo
  /**
  * Save current file, asking user for new destination name.
  * Report to statuBar.
  * @return false means user cancelled the SaveAs
  */
 boolean saveAsFile() {
  // Filer makes use of a java.awt.FileDialog, so its
  // mode property uses the same values as those of FileDialog.
  filer1.setMode(FileDialog.SAVE); // Use the SAVE version of the dialog.
  // Make the dialog visible as a modal (default) dialog box.
  filer1.show();
  // Upon return, getFile() will be null if user cancelled the dialog.
  if (filer1.getFile() != null) {
   // Non-null file property after return implies user
   // selected a filename to save to.
   // Set the current file name to the user's selection,
   // then do a regular saveFile.
   currFileName = filer1.getDirectory()+filer1.getFile();
   return saveFile();
  }
  else {
   return false;
  }
 }

//Metodo para abrir el archivo de texto.
 void AbrirArchivo(String NombreArchivo){
RBAuxII.RuleList =new Hashtable();
RBAuxII.ruleList= new Vector();
RBAuxII.goalClauseStack = new Stack();
RuleVariable VarRR;//= new RuleVariable("VarRR");
String Linea="";
String Clave;
String NV;
Rule ReglaX;
choice1.removeAll();
BCText.setText("");
  try{              //try
    LineNumberInputStream  SS;
    SS = new LineNumberInputStream(new FileInputStream(NombreArchivo));
    DataInputStream S= new DataInputStream(SS);
    while((Linea = S.readLine()) !=null) {   //while1
    choice1.addItem(Linea);
    StringTokenizer Line = new StringTokenizer(Linea,";");
      while(Line.hasMoreTokens()){        //while2
          Clavc=(String)Line.nextToken();
          BCText.append("\n Clave:"+ Clave);
          if(Clave.equals("<nombre>")){
                RBAuxII.name=filer1.getFile();
                NV= (String)Line.nextToken();
                BCText.append("\nBCNombre:"+RBAuxII.name);
                BCSelBC.addItem(RBAuxII.name);
                ChSelBC.addItem(RBAuxII.name);
                SelBCApplet.addItem(RBAuxII.name);
```

```
                    }
            if(Clave.equals("<variable>")){
                    VarRR = new RuleVariable(Line.nextToken());
                    VarRR.setPromptText((String)Line.nextToken());
                    VarRR.setLabels((String)Line.nextToken());
                    RBAuxII.variableList.put(VarRR.name,VarRR);
                    BCVarList.addItem(VarRR.name);

BCText.append("\nVName:"+VarRR.name+"\nPromp`:"+VarRR.promptText+"\nValues:"+VarRR.get
Labels());

                    }
            if(Clave.equals("<rule>")){
                    String NomRule=(String)Line.nextToken();
                    ContClause= Line.countTokens();
                    RuleVariable VarRule;
                    BCText.append("\nRuleBAse: "+RBAuxII.name);
                    BCText.append("\nRulename: "+NomRule);
                    BCText.append("\nNo Clauses:"+ContClause);

                    switch(ContClause){
                    case 2:
                    Clause1 = TransClause(Line.nextToken());
                    Clause2 = TransClause(Line.nextToken());
                    ReglaX = new Rule(RBAuxII,NomRule,Clause1,Clause2);
                    ReglaX.name=NomRule;
                    RBAuxII.RuleList.put(ReglaX.name,ReglaX);
                    break;
                    case 3:
                    Clause1 = TransClause(Line.nextToken());
                    Clause2 = TransClause(Line.nextToken());
                    Clause3 = TransClause(Line.nextToken());
                    ReglaX = new Rule(RBAuxII,NomRule,Clause1,Clause2
                                ,Clause3);
                    ReglaX.name=NomRule;
                    RBAuxII.RuleList.put(ReglaX.name,ReglaX);
                    break;
                    case 4:
                    Clause1 = TransClause(Line.nextToken());
                    Clause2 = TransClause(Line.nextToken());
                    Clause3 = TransClause(Line.nextToken());
                    Clause4 = TransClause(Line.nextToken());
                    ReglaX = new Rule(RBAuxII,NomRule,Clause1,Clause2
                                ,Clause3,Clause4);
                    ReglaX.name=NomRule;
                    RBAuxII.RuleList.put(ReglaX.name,ReglaX);
                    break;
                    case 5:
                    Clause1 = TransClause(Line.nextToken());
                    Clause2 = TransClause(Line.nextToken());
                    Clause3 = TransClause(Line.nextToken());
                    Clause4 = TransClause(Line.nextToken());
                    Clause5 = TransClause(Line.nextToken());
                    ReglaX = new Rule(RBAuxII,NomRule,Clause1,Clause2
                                ,Clause3,Clause4
                                ,Clause5);
```

```
ReglaX.name=NomRule;
RBAuxII.RuleList.put(ReglaX.name,ReglaX);
break;
case 6:
Clause1 = TransClause(Line.nextToken());
Clause2 = TransClause(Line.nextToken());
Clause3 = TransClause(Line.nextToken());
Clause4 = TransClause(Line.nextToken());
Clause5 = TransClause(Line.nextToken());
Clause6 = TransClause(Line.nextToken());
ReglaX = new Rule(RBAuxII,NomRule,Clause1,Clause2
                  ,Clause3,Clause4
                  ,Clause5,Clause6);
ReglaX.name=NomRule;
RBAuxII.RuleList.put(ReglaX.name,ReglaX);
break;
case 7:
Clause1 = TransClause(Line.nextToken());
Clause2 = TransClause(Line.nextToken());
Clause3 = TransClause(Line.nextToken());
Clause4 = TransClause(Line.nextToken());
Clause5 = TransClause(Line.nextToken());
Clause6 = TransClause(Line.nextToken());
Clause7 = TransClause(Line.nextToken());
ReglaX = new Rule(RBAuxII,NomRule,Clause1,Clause2
                  ,Clause3,Clause4
                  ,Clause5,Clause6
                  ,Clause7);
ReglaX.name=NomRule;
RBAuxII.RuleList.put(ReglaX.name,ReglaX);
break;
case 8:
Clause1 = TransClause(Line.nextToken());
Clause2 = TransClause(Line.nextToken());
Clause3 = TransClause(Line.nextToken());
Clause4 = TransClause(Line.nextToken());
Clause5 = TransClause(Line.nextToken());
Clause6 = TransClause(Line.nextToken());
Clause7 = TransClause(Line.nextToken());
Clause8 = TransClause(Line.nextToken());
ReglaX = new Rule(RBAuxII,NomRule,Clause1,Clause2
                  ,Clause3,Clause4
                  ,Clause5,Clause6
                  ,Clause7,Clause8);
ReglaX.name=NomRule;
RBAuxII.RuleList.put(ReglaX.name,ReglaX);
break;
case 9:
Clause1 = TransClause(Line.nextToken());
Clause2 = TransClause(Line.nextToken());
Clause3 = TransClause(Line.nextToken());
Clause4 = TransClause(Line.nextToken());
Clause5 = TransClause(Line.nextToken());
Clause6 = TransClause(Line.nextToken());
Clause7 = TransClause(Line.nextToken());
Clause8 = TransClause(Line.nextToken());
```

```
                    Clause9 = TransClause(Line.nextToken());
                    ReglaX = new Rule(RBAuxII,NomRule,Clause1,Clause2
                                ,Clause3,Clause4
                                ,Clause5,Clause6
                                ,Clause7,Clause8
                                ,Clause9);
                    ReglaX.name=NomRule;
                    RBAuxII.RuleList.put(ReglaX.name,ReglaX);
                    break;
                    case 10:
                    Clause1 = TransClause(Line.nextToken());
                    Clause2 = TransClause(Line.nextToken());
                    Clause3 = TransClause(Line.nextToken());
                    Clause4 = TransClause(Line.nextToken());
                    Clause5 = TransClause(Line.nextToken());
                    Clause6 = TransClause(Line.nextToken());
                    Clause7 = TransClause(Line.nextToken());
                    Clause8 = TransClause(Line.nextToken());
                    Clause9 = TransClause(Line.nextToken());
                    Clause10 = TransClause(Line.nextToken());
                    ReglaX = new Rule(RBAuxII,NomRule,Clause1,Clause2
                                ,Clause3,Clause4
                                ,Clause5,Clause6
                                ,Clause7,Clause8
                                ,Clause9,Clause10);
                    ReglaX.name=NomRule;
                    RBAuxII.RuleList.put(ReglaX.name,ReglaX);
                    break;
                            }
                    BCText.append("\n----------");
                    }
          if(Clave.equals("<msgin>")){
                    RBAuxII.MsgIntro=(String)Line.nextToken();

          }
          if(Clave.equals("<msgg>")){
                    RBAuxII.MsgGen=(String)Line.nextToken();

          }
          if(Clave.equals("<msgf>")){
                    RBAuxII.MsgFinal=(String)Line.nextToken();

          }



          if(Clave.equals("<BCFIN>")){
            BasesCon.put(RBAuxII.name,RBAuxII);
            BCText.append("SE Agrego la base de Conocimiento:"+RBAuxII.name +
Line.nextToken());
            ContClause=1;
            textArea2.append("\n File:"+currFileName);

                    }
        } //while2
      } //while1
```

```
      this.currFileName =NombreArchivo;
      this.dirty= false;

      } //try

  catch(IOException e)
  {
   BCText.setText("Error al abrir archivo: " + NombreArchivo);
  }
  }

void GuardaBC_actionPerformed(ActionEvent e) {
    GuardaBC.enable(false);
    saveFile();
 }

 void CargaBC_actionPerformed(ActionEvent e) {
  BCVarList.removeAll();
  BCListReg.removeAll();
  filer1.setMode(FileDialog.LOAD);
  filer1.show();
  String Beta = "NO";
  for (int k=0;k<BCSelBC.getItemCount();k++){
      if(filer1.getFile().equals(BCSelBC.getItem(k))){
       MsgError.MsgErr.setText("Ya hay una Base de Reglas \""+filer1.getFile()+"\" cargada");
       MsgError.show();
       Beta="SI";}
  }
  BCText.append("\nBeta= "+Beta);
   if(filer1.getFile() != null) {
   if(Beta.equals("NO")){AbrirArchivo(filer1.getDirectory() + filer1.getFile());
     currFileName = filer1.getFile();}
     }
// textArea2.append("\n File:"+currFileName);

}
//Metodo de transformacion de la cadena ascci en una Clause
Clause TransClause(String ClauseX){
RuleVariable RVar;
String Valor=RBAuxII.name;
Clause ClauseZ;
StringTokenizer ClauseY = new StringTokenizer(ClauseX,"&");
String Alfa= ClauseY.nextToken();
RVar = (RuleVariable)RBAuxII.variableList.get(Alfa);
Valor = RVar.name;
Condition COND = new Condition(ClauseY.nextToken());
Valor = ClauseY.nextToken();
ClauseZ = new Clause(RVar,COND,Valor);
return ClauseZ;

}

void AppletGenApplet() {
textArea2.setText("");
String BCNA = SelBCApplet.getSelectedItem();
RBAux =(RuleBase)BasesCon.get(BCNA);
```

```
RBAux.reset();
textArea2.append("import java.applet.*;\n");
textArea2.append("import java.awt.*;\n");
textArea2.append("import java.awt.event.*;\n");
textArea2.append("import java.util.*;\n");
textArea2.append("import Dialog1;\n");
textArea2.append("public class "+BCNA+"Applet extends Applet {\n");

// carga y despliega inicio de archivo
String Linea;
  try{   //(1)
    LineNumberInputStream  INIC;
    INIC = new LineNumberInputStream(new FileInputStream("C:/RW/Inic.txt"));
    DataInputStream INIC2= new DataInputStream(INIC);
    while((Linea = INIC2.readLine()) !=null) {   //while1
    textArea2.append(Linea+"\n");
                    }

    textArea2.append("\n   public "+BCNA +"Applet() {\n");
    LineNumberInputStream SIG;
    SIG = new LineNumberInputStream(new FileInputStream("C:/RW/Sig.txt"));
    DataInputStream SIG2= new DataInputStream(SIG);
    while((Linea = SIG2.readLine()) !=null) {   //while1
    textArea2.append(Linea+"\n");
                    }

//genera codigo de base de conocimiento  Variables y Reglas
    textArea2.append("BaseReg = new RuleBase(\""+RBAux.name +"\");\n");
    textArea2.append("BaseReg.ruleList = new Vector();\n");
    textArea2.append("BaseReg.goalClauseStack = new Stack();\n");
    textArea2.append("BaseReg.variableList = new Hashtable();\n");
    textArea2.append("BaseReg.RuleList = new Hashtable();\n");
    textArea2.append("\nCondition Igual = new Condition(\"=\");\n");
    textArea2.append("Condition Difer = new Condition(\"!=\");\n");
    textArea2.append("Condition Menorque = new Condition(\"<\");\n");
    textArea2.append("Condition Mayorque = new Condition(\">\");\n");

     Enumeration Vars = RBAux.variableList.elements();
     RuleVariable RVarApp;
     while (Vars.hasMoreElements()) {
        RVarApp = (RuleVariable)Vars.nextElement();
    textArea2.append("RuleVariable     "+RVarApp.name     +"     =     new     RuleVariable
(\""+RVarApp.name+"\");\n");
    textArea2.append(RVarApp.name+".setLabels(new String(\""+RVarApp.getLabels()+"\"));\n");
    textArea2.append(RVarApp.name+".setPromptText(new String(\"" + RVarApp.promptText +
"\"));\n");
    textArea2.append("BaseReg.variableList.put(" +RVarApp.name+".name," + RVarApp.name +
");\n");
    textArea2.append("choice2.addItem("+RVarApp.name+".name);\n");
    textArea2.append("choice4.addItem("+RVarApp.name+".name);\n\n");
                    }
     Vars = RBAux.RuleList.elements();
     Rule RegX;
      while (Vars.hasMoreElements()){
      RegX = (Rule)Vars.nextElement();
      RegX.DispApp(textArea2);
```

```
                    }

 // Despliega la cola del archivo del applet
        LineNumberInputStream COLA;
        COLA = new LineNumberInputStream(new FileInputStream("C:/RW/cola.txt"));
        DataInputStream COLA2= new DataInputStream(COLA);
        while((Linea = COLA2.readLine()) !=null) {   //while1
        textArea2.append(Linea+"\n");
                    }
        }  // se cierra el try (1)

 catch(IOException Leer)
 {
  textArea2.setText("Error no se encontro codigo de inicio: Inic.txt");
 }

 //Guarda archivos necesarios para el applet
  try{  //(2)
     File RApplet = new File("C:/RW/"+BCNA+"Applet.java");
     FileWriter Out = new FileWriter(RApplet);
     String Codigo = textArea2.getText();
     Out.write(Codigo);
     Out.close();

 } //Cierra try(2)
 catch (IOException Escribe){
    textArea2.append("\n\n\n Error al escribir en el archivo: "Applet"");
    }

 // Manda comando de compilacion

 // Genera archivos de instalacion
}

 void OpList_actionPerformed(ActionEvent e) {
      VarOK.setLabel("Actualizar");
   valores= "";
   String Vaux = VarList.getSelectedItem();
   RBAux= (RuleBase)BasesCon.get(this.getTitle());
   RuleVariable RVX = (RuleVariable)RBAux.variableList.get(Vaux);
   NomVartxt.setText(RVX.name);
   ProVartxt.setText(RVX.promptText);
   OpVartxt.setText("");
   OpList.removeAll();
      Enumeration Vals = RVX.labels.elements();
       while (Vals.hasMoreElements()){
       Vaux= (String)Vals.nextElement();
       if (valores == ""){ valores = Vaux;}
       else {valores = valores+ " " + Vaux;}
         OpList.addItem(Vaux);
         }
}

void CRBEdClause_actionPerformed(ActionEvent e) {
 }
```

```
void SelValAnt_itemStateChanged(ItemEvent e){
}

 void ChSelMeta_itemStateChanged(ItemEvent e) {
 }

void tabsetPanel1_actionPerformed(ActionEvent e) {
//   CRBAND.enable(alse);
   CRBAND.disable();
 }

 void NomRegTxt_keyTyped(KeyEvent e) {
  if (ValConsec = true){
     CRBOK.enable(); }
 }

 void button9_actionPerformed(ActionEvent e) {
  button10.enable(false);
  BCVarList.removeAll();
  BCListReg.removeAll();
  RBAux = (RuleBase)BasesCon.get(BCSelBC.getSelectedItem());
//  BCText.setText(RBAux.MsgIntro);
  if (button9.getLabel() == "Msg Intro"){
  BCText.setText(RBAux.MsgIntro);
        button9.setLabel("OK");
        GuardaBC.setEnabled(false);
        }

  else {
         button9.setLabel("Msg Intro");
         RBAux.MsgIntro=BCText.getText();
         button10.enable(true);
         }
//  BCText.setText(MsgIntro);
 }

 void button10_actionPerformed(ActionEvent e) {
  button9.enable(false);
  BCVarList.removeAll();
  BCListReg.removeAll();
  RBAux = (RuleBase)BasesCon.get(BCSelBC.getSelectedItem());
//  BCText.setText(RBAux.MsgIntro);
  if (button10.getLabel() == "Msg Final"){
  BCText.setText(RBAux.MsgFinal);
        button10.setLabel("OK");
        GuardaBC.setEnabled(false);
        }

  else {
         button10.setLabel("Msg Final");
         RBAux.MsgFinal=BCText.getText();
         button9.enable(true);
         }
}

void CFGVar_itemStateChanged(ItemEvent e) {
```

```
    RBAux= (RuleBase)BasesCon.get(this.getTitle());
    String CFVars = CFGVar.getSelectedItem();
    CFGValList.removeAll();
    RuleVariable CFVar = (RuleVariable)RBAux.variableList.get(CFVars);
    Enumeration CFValss = CFVar.labels.elements();
    while (CFValss.hasMoreElements()) {
        CFGValList.addItem(((String)CFValss.nextElement()));
        }

  }

  void GETURL_actionPerformed(ActionEvent e) {
 Image img= TIMAGEN.getImage();
    try{
     IMGN2.setImageName("WX4\\Imagenes\\car8.jpg");
}
     catch (Exception ex) {
    TxtURL.setText("Error al leer imagen");
   }
}

  void OpVartxt_keyTyped(KeyEvent e) {
}

  void CGFAppList_actionPerformed(ActionEvent e) {
       Image img= TIMAGEN.getImage();
   try{
     IMGN2.setImageName("WX4\\Imagenes\\"+CGFAppList.getSelectedItem()+".jpg");
}
     catch (Exception eimg) {
    TxtURL.setText("Error al leer imagen "+CGFAppList.getSelectedItem());
   }
  }
  public String PideNombre(){
//*********************************************************
  Frame NomBasee= new Frame("Nueva Base de Reglas");
  NOMBOX = new Nombre(frame);
  NOMBOX.setModal(true);
  NOMBOX.setLocation(250,250);
  NOMBOX.show();
   String Resp = NOMBOX.NomTxt.getText();
//*********************************************************
   if (Resp == "") {Resp="BaseReg1";}
   return Resp;}

 void button4_actionPerformed(ActionEvent e) {
  String NomBC = this.PideNombre();
  RBNueva = new RuleBase(NomBC);
 // Transporte = new RuleBase("Autos");
  RBNueva.ruleList = new Vector();
  RBNueva.goalClauseStack = new Stack();
  RBNueva.variableList = new Hashtable();
  RBNueva.RuleList = new Hashtable();
  BasesCon.put(RBNueva.name,RBNueva);
  this.setTitle(RBNueva.name);
```

```
    BCText.setText(NomBC);
    BCSelBC.addItem(NomBC);
    BCSelBC.select(NomBC);
    ChSelBC.addItem(NomBC);
    ChSelBC.select(NomBC);
    BCVarList.removeAll();
    BCListReg.removeAll();
    BCVarList.addItem("Vacia");
    BCListReg.addItem("Vacia");
    ChSelVar.removeAll();
    ChSelVal.removeAll();
    ChSelMeta.removeAll();
    textArea3.setText("");
    SelVarAnt.removeAll();
    SelVarCons.removeAll();
    SelValAnt.removeAll();
    SelValCons.removeAll();
    RegList.removeAll();
    list3.removeAll();
    textArea1.setText("");
    NomRegTxt.setText("");
    BCActual.setText(NomBC);
    VarList.removeAll();
    OpList.removeAll();
    NomVartxt.setText("");
    ProVartxt.setText("");
    OpVartxt.setText("");
    VAR.requestFocus();
  }

  void GenApplet_actionPerformed(ActionEvent e) {
    saveApplet();
  }
/* Metodo que guarda base de conocimiento en formato texto
 ----------------------------------------------------------------*/
   boolean saveApplet()
   {
    // Handle the case where we don't have a file name yet.
    if (currAppletName == null) {
     return saveAsApplet();
    }
    try
    {
     // Open a file of the current name.
     File file = new File (currAppletName);
     // Create an output writer that will write to that file.
     // FileWriter handles international characters encoding conversions.
     FileWriter out = new FileWriter(file);
     String text =textArea2.getText();
     out.write(text);
     out.close();
     this.dirty = false;
     //updateCaption();
     statusBar.setText("Archivo" + currAppletName + " Se guardo correctamente");
     return true;
    }
```

```
    catch (IOException e) {
      statusBar.setText("Error al guardar "+currAppletName);
    }
    return false;
  }
 // fin metodo salvar archivo
boolean saveAsApplet() {
    // Filer makes use of a java.awt.FileDialog, so its
    // mode property uses the same values as those of FileDialog.
    filer1.setMode(FileDialog.SAVE); // Use the SAVE version of the dialog.
    // Make the dialog visible as a modal (default) dialog box.
    filer1.show();
    // Upon return, getFile() will be null if user cancelled the dialog.
    if (filer1.getFile() != null) {
      // Non-null file property after return implies user
      // selected a filename to save to.
      // Set the current file name to the user's selection,
      // then do a regular saveFile.
      currAppletName = filer1.getDirectory()+"BC5"; //+filer1.getFile();
//      textArea2.setText(currAppletName);
      return saveApplet();
    }
    else {
      return false;
    }
  }
} // fin de Aplicacion

/* Clase Frame1_AboutBox, es un cuadro de dialogo para información del sistema.*/
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import borland.jbcl.control.BevelPanel;
import borland.jbcl.control.ImageControl;
public class Frame1_AboutBox extends Dialog implements ActionListener{
  BevelPanel panel1 = new BevelPanel();
  BevelPanel panel2 = new BevelPanel();
  BevelPanel insetsPanel1 = new BevelPanel();
  BevelPanel insetsPanel2 = new BevelPanel();
  BevelPanel insetsPanel3 = new BevelPanel();
  JButton button1 = new JButton();
  ImageControl imageControl1 = new ImageControl();
  JLabel label1 = new JLabel();
  JLabel label2 = new JLabel();
  JLabel label3 = new JLabel();
  JLabel label4 = new JLabel();
  BorderLayout borderLayout1 = new BorderLayout();
  BorderLayout borderLayout2 = new BorderLayout();
  FlowLayout flowLayout1 = new FlowLayout();
  FlowLayout flowLayout2 = new FlowLayout();
  GridLayout gridLayout1 = new GridLayout();
  String product = "WebXpert V1.0";
  String version = "Lab. de Sistemas Inteligentes";
  String copyright = "Centro de Instrumentos UNAM";
  String comments = "Your description";
```

```
 public Frame1_AboutBox(Frame parent) {
  super(parent);
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
   jbInit();}
  catch (Exception e) {
   e.printStackTrace();}
  pack();
 }

 private void jbInit() throws Exception {
  this.setTitle("Acerca de WebXpert");
  setResizable(false);
  panel1.setLayout(borderLayout1);
  panel2.setLayout(borderLayout2);
  insetsPanel1.setLayout(flowLayout1);
  insetsPanel1.setBevelInner(BevelPanel.FLAT);
  insetsPanel2.setLayout(flowLayout1);
  insetsPanel2.setMargins(new Insets(10, 10, 10, 10));
  insetsPanel2.setBevelInner(BevelPanel.FLAT);
  gridLayout1.setRows(4);
  gridLayout1.setColumns(1);
  label1.setText("WebXpert V1.0");
  label2.setText(version);
  label3.setText("Centro de Instrumentos UNAM");
  label4.setText("Desarrollo de Sistemas Expertos para Web");
  insetsPanel3.setLayout(gridLayout1);
  insetsPanel3.setMargins(new Insets(10, 60, 10, 10));
  insetsPanel3.setBevelInner(BevelPanel.FLAT);
  button1.setText("OK");
  //imageControl1.setImageName("WX4\\jb2box_std.jpg");
  button1.addActionListener(this);
  insetsPanel2.add(imageControl1, null);
  panel2.add(insetsPanel2, BorderLayout.WEST);
  this.add(panel1, null);
  insetsPanel3.add(label1, null);
  insetsPanel3.add(label2, null);
  insetsPanel3.add(label3, null);
  insetsPanel3.add(label4, null);
  panel2.add(insetsPanel3, BorderLayout.CENTER);
  insetsPanel1.add(button1, null);
  panel1.add(insetsPanel1, BorderLayout.SOUTH);
  panel1.add(panel2, BorderLayout.NORTH);
 }
 protected void processWindowEvent(WindowEvent e) {
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
   cancel();}
  super.processWindowEvent(e);
 }
 void cancel() {
  dispose();}
 public void actionPerformed(ActionEvent e) {
  if (e.getSource() == button1) {
   cancel();}
 }
}
```

Apéndice B.

Uso del sistema para implementar un sistema experto

Para entender el uso de la interfaz, utilizaremos el ejemplo siguiente de un sistema experto para asesorar a los estudiantes de una Universidad en la selección de estudios relacionados con áreas de la computación.

Los expertos en el área, han determinado que las siguientes variables son importantes, para saber si un estudiante tiene cualidades para realizar estudios relacionados con alguna área de la computación, estas variables son:

Gusto por las computadoras.
Gusto por las matemáticas.
Habilidad para resolver problemas.
Destreza manual.
Lugar de trabajo.
Solvencia económica al terminar los estudios.
Resolver problemas.
Nivel de aptitud hacia la computación
Nivel de interés hacia la computación

Al medir estas variables, los expertos han establecido tres parámetros para determinar si el estudiante debe realizar estudios relacionados con alguna área de la computación, ó no debe hacerlo, estos dos parámetros son: Nivel de Aptitud hacia la computación, Nivel de Interés hacia la computación y si desea tener solvencia económica al terminar los estudios.

Con base a la medición de las variables antes descritas, se tiene la siguiente tabla de decisión para hacer recomendaciones al estudiante; dependiendo del nivel de aptitud y el nivel de interés que éste tenga hacia la computación.

Tabla1

Regla	Nivel Aptitud	Nivel Interés	Necesidad Financiera	Recomendación
1-1	Nivel 0	-	-	Area diferente de computación
1-2	-	Nivel 0	-	Area diferente de computación
1-3	Nivel 1	Nivel 1	-	Técnico en Computación
1-4	Nivel 1	Nivel 2	Si	Programador de Computadoras
1-5	Nivel 1	Nivel 2	No	Ciencias de la Computación
1-6	Nivel 1	Nivel 3	Si	Programador de Computadoras
1-7	Nivel 1	Nivel 3	No	Ciencias de la Computación
1-8	Nivel 2	Nivel 1	Si	Programador de Computadoras
1-9	Nivel 2	Nivel 1	No	Ciencias de la Computación
1-10	Nivel 2	Nivel 2	Si	Programador de Computadoras
1-11	Nivel 2	Nivel 2	No	Ciencias de la Computación
1-12	Nivel 2	Nivel 3	Si	Programador de Computadoras
1-13	Nivel 2	Nivel 3	No	Ciencias de la Computación
1-14	Nivel 3	Nivel 1	-	Técnico en Computación

1-15	Nivel 3	Nivel 2	-	Técnico en Computación
1-16	Nivel 3	Nivel 3	-	**Técnico en Computación**
1-17	Nivel 4	Nivel 1	-	Técnico en Computación
1-18	Nivel 4	Nivel 2	-	Programador de Computadoras
1-19	Nivel 4	Nivel 3	-	Programador de Computadoras
1-20	Nivel 5	Nivel 1	-	Programador de Computadoras
1-21	Nivel 5	Nivel 2	-	Programador de Computadoras
1-22	Nivel 5	Nivel 3	-	Programador de Computadoras

Para determinar el nivel de aptitud, lo han hecho con las variables*: Aptitud para las matemáticas, Aptitud para la programación* y la *Destreza manual*, con lo cual se ha establecido la siguiente tabla de decisión.

Tabla2

Regla	Aptitud Matemática	Aptitud de programación	Destreza Manual	Nivel de Aptitud
2-1	Si	Si	Si	Nivel 1
2-2	Si	Si	No	Nivel 2
2-3	Si	No	Si	Nivel 3
2-4	Si	No	No	Nivel 0
2-5	No	Si	Si	Nivel 4
2-6	No	Si	No	Nivel 5
2-7	No	No	Si	Nivel 3
2-8	No	No	No	Nivel 0

De igual manera, para medir el nivel de interés se usaron las variables: *Gusto por las computadoras, Habilidad para resolver problemas, Lugar de trabajo y sí le gusta reparar cosas*, con lo cual se tiene la siguiente tabla de decisión.

Tabla3

Regla	Computadoras	Resolver problemas	Lugar de trabajo	Reparar Cosas	Nivel de interés
3-1	Si	Si	Oficina	Si	Nivel 3
3-2	Si	Si	-	No	Nivel 2
3-3	Si	Si	campo	Si	Nivel 1
3-4	Si	No	-	-	Nivel 0
3-5	No	-	-	-	Nivel 0

Para implementar el sistema experto, debemos establecer las variables de la base de conocimiento y sus valores validos, así las variables las definimos como:

Gmate: Gusto por las matemáticas.
ApProg: Aptitud de programación.
DesManual: Destreza manual.
NivAptitud: Nivel de aptitud hacia la computación
GComp: Gusto por las computadoras.
ResProb Habilidad para resolver problemas.
LugTrab: Lugar de trabajo.

RepCos: Te gusta reparar cosas
LugTrab: Lugar de trabajo.
SolEcono: Solvencia económica al terminar los estudios.
Recom Recomendación final

El conjunto de reglas que servirá para determinar la recomendación que el sistema dará al estudiante como resultado final, se construyen con la tabla de decisión Tabla1:

REGLA1-1: Si NivAptitud=Nivel 0
Ent- Recom= Otra área no relacionadas con la computación

REGLA1-2: Si NivInteres=Nivel 0
Ent- Recom= Otra área no relacionadas con la computación

REGLA1-3: Si NivAptitud=Nivel 1
y NivInteres=Nivel 1
Ent- Recom= Técnico en computación

REGLA1-4: Si NivAptitud=Nivel 1
y NivInteres=Nivel 2
y SolEcono=Si
Ent- Recom=Programador de computadoras

REGLA1-5A: Si NivAptitud=Nivel 1
y NivInteres=Nivel 2
y SolEcono=No
Ent- Recom=Ciencias de la Computación

REGLA1-6: Si NivAptitud=Nivel 1
y NivInteres=Nivel 3
y SolEcono=Si
Ent- Recom=Programador de computadoras

REGLA1-7: Si NivAptitud=Nivel 1
y NivInteres=Nivel 3
y SolEcono=No
Ent- Recom=Ciencias de la Computación

REGLA1-8: Si NivAptitud=Nivel 2
y NivInteres=Nivel 1
y SolEcono=Si
Ent- Recom=Programador de computadoras

REGLA1-9: Si NivAptitud=Nivel 2
y NivInteres=Nivel 1
y SolEcono=No
Ent- Recom=Ciencias de la Computación

REGLA1-10: Si NivAptitud=Nivel 2
y NivInteres=Nivel 2
y SolEcono=Si
Ent- Recom=Programador de computadoras

REGLA1-11: Si NivAptitud=Nivel 2
y NivInteres=Nivel 2
y SolEcono=No
Ent- Recom=Ciencias de la Computación

REGLA1-12: Si NivAptitud=Nivel 2
y NivInteres=Nivel 3
y SolEcono=Si
Ent- Recom=Programador de computadoras

REGLA1-13: Si NivAptitud=Nivel 2
y NivInteres=Nivel 3
y SolEcono=No
Ent- Recom=Técnico en computación

REGLA1-14: Si NivAptitud=Nivel 3
y NivInteres=Nivel 1
Ent- Recom=Técnico en computación

REGLA1-15: Si NivAptitud=Nivel 3
y NivInteres=Nivel 2
Ent- Recom=Técnico en computación

REGLA1-16: Si NivAptitud=Nivel 3
y NivInteres=Nivel 3
Ent- Recom=Técnico en computación

REGLA1-17: Si NivAptitud=Nivel 4
y NivInteres=Nivel 1
Ent- Recom=Técnico en computación

REGLA1-18: Si NivAptitud=Nivel 4
y NivInteres=Nivel 2
Ent- Recom=Programador de computadoras

REGLA1-19: Si NivAptitud=Nivel 4
y NivInteres=Nivel 3
Ent- Recom=Programador de computadoras

REGAL1-20: Si NivAptitud=Nivel 5
y NivInteres=Nivel 1
Ent- Recom=Programador de computadoras

REGLA1-21: Si NivAptitud=Nivel 5
y NivInteres=Nivel 2
Ent- Recom=Programador de computadoras

REGLA1-22: Si NivAptitud=Nivel 5
y NivInteres=Nivel 3
Ent- Recom=Programador de computadoras

El conjunto de reglas, que servirá para determinar el nivel de aptitud hacia la computación que tiene el estudiante, se construye con la tabla de decisión Tabla2:

REGLA2-1: Si GMate=Si
y ApProg=Si
y DesManual=Si
Ent- NivAptitud=Nivel 1

REGLA2-2: Si GMate=Si
y ApProg=Si
y DesManual=No
Ent- NivAptitud=Nivel 2

REGLA2-3: Si GMate=Si
y ApProg=No
y DesManual=Si
Ent- NivAptitud=Nivel 3

REGLA2-4: Si GMate=Si
y ApProg=No
y DesManual=No
Ent- NivAptitud=Nivel 0

REGLA2-5: Si GMate=No
y ApProg=Si
y DesManual=Si
Ent- NivAptitud=Nivel 4

REGLA2-6: Si GMate=No
y ApProg=Si
y DesManual=No
Ent- NivAptitud=Nivel 5

REGLA2-7: Si GMate=No
y ApProg=No
y DesManual=Si
Ent- NivAptitud=Nivel 3

REGLA2-8: Si GMate=No
y ApProg=No
Ent- NivAptitud=Nivel 0

El conjunto de reglas que servirá para determinar el nivel de interés hacia la computación que tienen el estudiante se construye con la tabla de decisión Tabla3:

REGLA3-1: Si GComp=Si
y ResProb=Si
y LugTrab=Una Oficina
y RepCos=Si
Ent- NivInteres=Nivel 3

REGLA3-2: Si GComp=Si
y ResProb=Si
y RepCos=No
Ent- NivInteres=Nivel 2

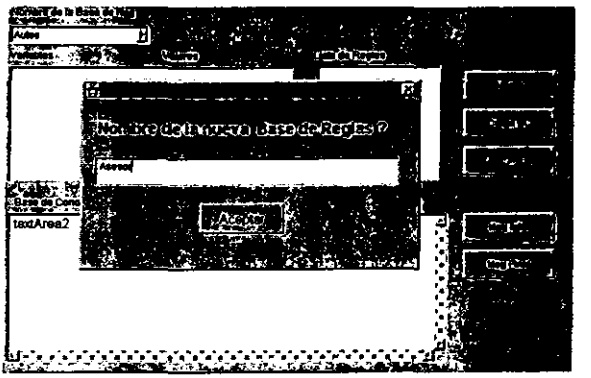
REGLA3-3: Si GComp=Si
y ResProb=Si
y LugTrab=De campo
y RepCos=Si
Ent- NivInteres=Nivel 1

REGLA3-4: Si GComp=Si
y ResProb=No
Ent- NivInteres=Nivel 0

REGLA3-5: Si GComp=No
Ent- NivInteres=Nivel 0

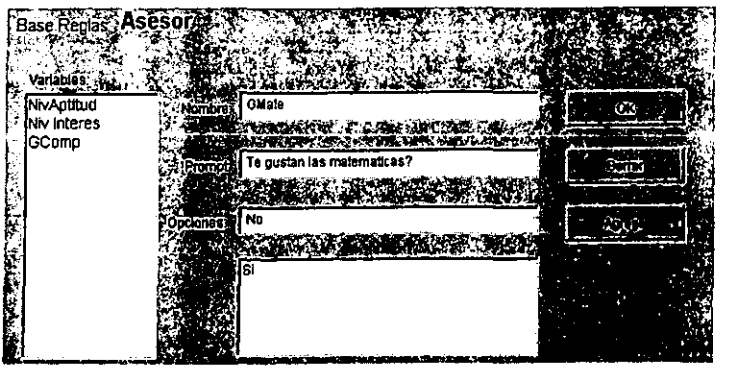
Para construir la base de conocimiento, primero declaramos una nueva base de reglas, para ello seleccionamos la carpeta "*Base de Conocimiento*" y hacemos clic en Nueva, así el sistema construye una nueva base de reglas en blanco y está lista para alimentarla con los datos.

Nueva Base de Reglas



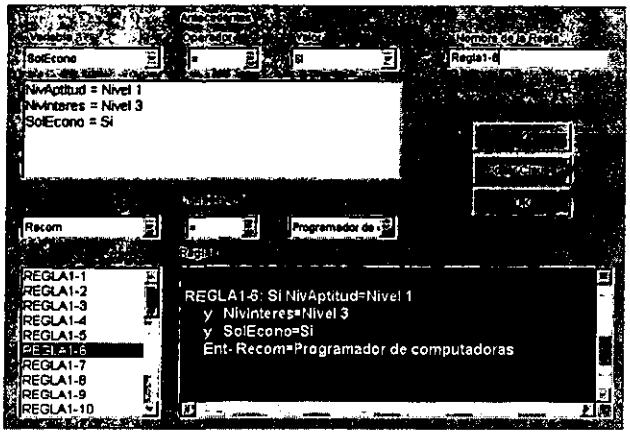
Ahora debemos primero declarar todas las variables de la Base de Reglas, para esto seleccionamos la carpeta de "*Variables*" y proporcionamos los datos necesarios para cada una de las variables: Nombre, prompt valores etc.

Declaración de variables.



Con todas las variables declaradas, podemos ahora construir todas las cláusulas y con éstas, las reglas; abra la carpeta de "*Cláusulas/Reglas*" y seleccione la variable, el operador y el valor valido que formaran a la cláusula; cada cláusula de las premisas es unida por la conjunción, esto se hace con el botón de control "Y"; de la misma manera se construye la cláusula consecuente; en el campo de nombre de la regla, se debe colocar un nombre que identifique a la regla y para finalizar, haga clic en el botón de control OK para agregar la nueva regla a la base de conocimiento.

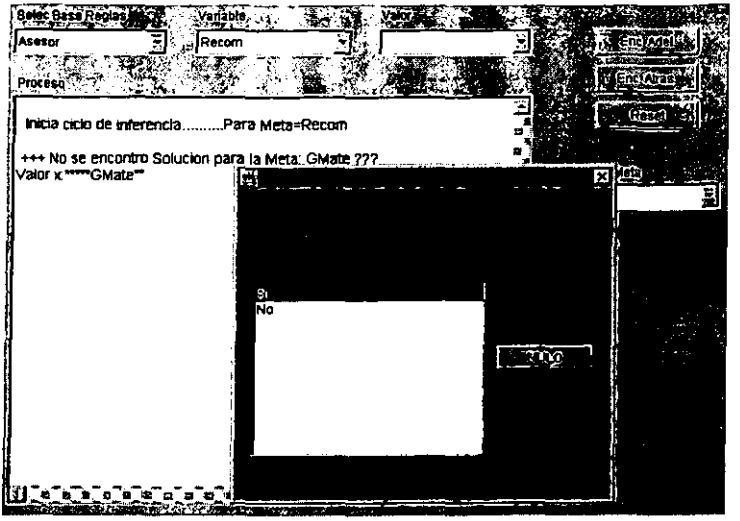
Definición de cláusulas y reglas

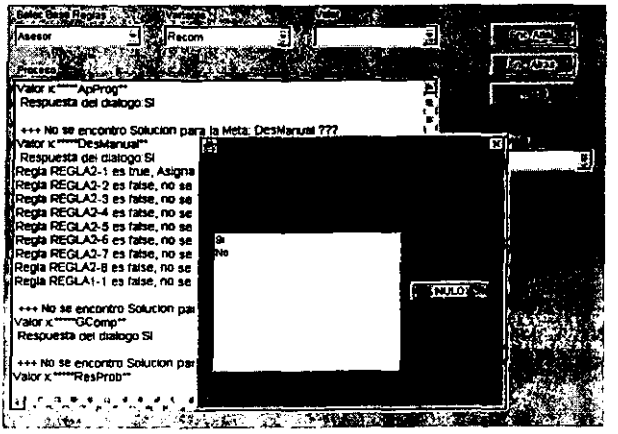


Una vez terminada la base de reglas, procedemos a realizar las pruebas. Abra la carpeta de Encadenamiento y en la casilla de selección, verifique que la base de reglas Asesor este seleccionada. Sólo se necesita una meta, que es la recomendación final por lo que al correr la máquina de inferencia con encadenamiento hacia atrás, el sistema realizara una serie de preguntas al usuario acerca de sus cualidades y habilidades; al final el sistema despliega una recomendación para el estudiante.

Para realizar las pruebas, con el botón de control "*Reset*", se inicializa la base de conocimiento asignando a todas las variables un valor nulo, en las casillas de selección de variables, puede asignar valores específicos a las variables. Con los botones de Control "*Enc. Adel.*" y "*Enc. Atrás*", se pueden aplicar procesos de inferencia para Encadenamiento hacia Adelante y Encadenamiento hacia Atrás respectivamente. Al iniciar los procesos de inferencia, en el área de texto se indican los procesos que se llevan a cabo, mucha información acerca del comportamiento del sistema experto es desplegada para fines de depuración y explicación. Con esta pantalla de explicación, el usuario puede observar detalladamente el funcionamiento de la máquina de inferencia y también el avance en la captura de la base de conocimiento.

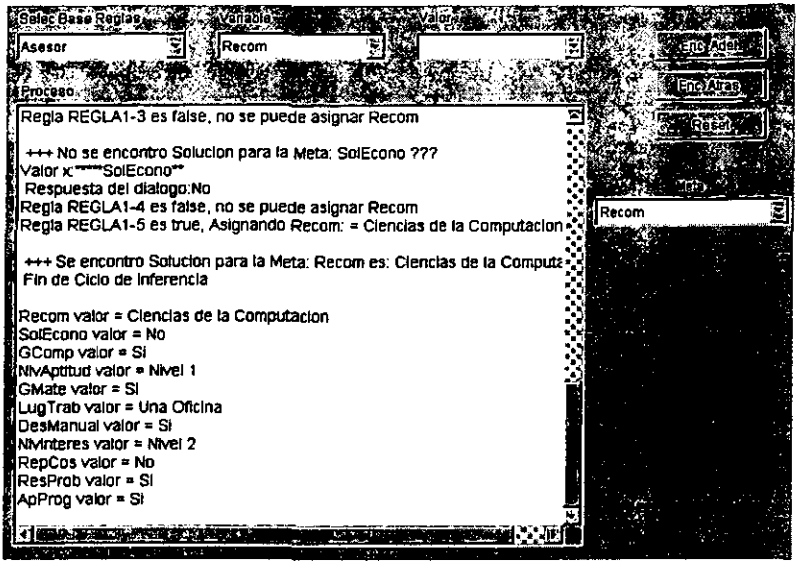
Procesos de inferencia en ambiente de desarrollo.





Al final del ciclo de inferencia aplicado, en la pantalla de explicación se presentan los resultados del ciclo de inferencia, aquí se pueden presentar tanto la recomendación final como los valores asignados a cada una de las variables durante el ciclo de inferencia aplicado; esto es para ambos, Encadenamiento hacia Adelante y Encadenamiento hacia atrás.

Resultados finales después del ciclo de inferencia.

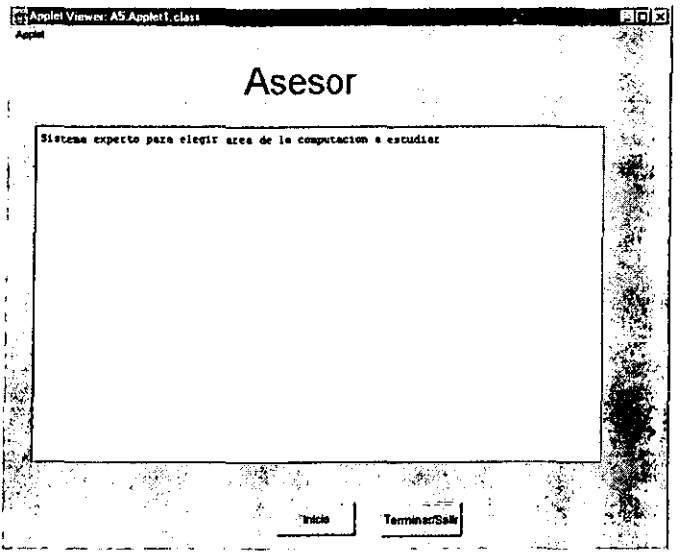


Ahora nuestro sistema experto esta listo para ser colocado en el servidor Web, para esto, abra la carpeta de opciones, seleccionamos la base de reglas que acabamos de terminar, y utilizando el botón de control "Genera Applet", el sistema pedirá al usuario que indique una ruta donde guardar los programas necesarios para correr en Web, como la computadora que tienen instalada esta aplicación, está conectada a nuestro servidor Web, podemos seleccionar directamente el directorio donde se encuentra el servicio Web, aquí se colocaran los archivos del nuevo sistema experto que corre en un Applet Java.

Una vez colocado el sistema experto en el servidor Web, la forma de acceso a él es a través de Internet para ello se puede realizar una consulta directa, entrando al navegador de Internet y haciendo la llamada a la siguiente dirección:

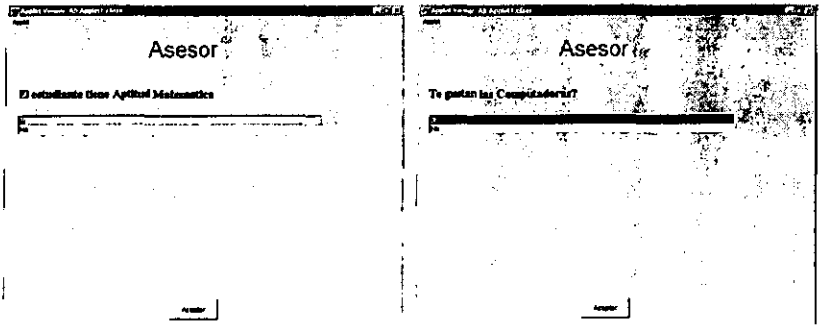
http://wizard.cinstrums.unam.mx/Asesor.html

Pantalla Applet Asesor pantalla de inicio:

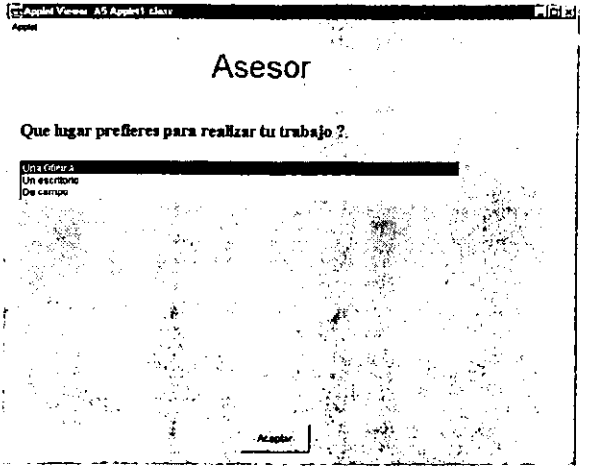


Al iniciar el sistema, el usuario empieza la consulta y el sistema experto Asesor realiza una serie de preguntas para determinar su nivel de interés y nivel de aptitud hacia las computadoras.

Sistema Experto Asesor montado en un Applet Java.



Interacción con el usuario.



Al final el sistema dará una recomendación que se infiere con los datos proporcionados por el usuario. La pantalla de explicación, también se presenta en este modo de operación. El sistema permite volver a correr el proceso de inferencia para una nueva consulta.

Resultado final.