12
2ej.

UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
A R A G O N

TEORIA Y APLICACIONES DE LOS SISTEMAS EXPERTOS

TESIS CON
FALLA DE ORIGEN

T E S I S

QUE PARA OBTENER EL TITULO DE:
INGENIERO EN COMPUTACION
P R E S E N T A :

AMILCAR AMADO MONTERROSA ESCOBAR

SAN JUAN DE ARAGON, ESTADO DE MEXICO 1992

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

INDICE

CAPITULO I: INTRODUCCION

CAPITULO II: ESTRUCTURA DE LOS SISTEMAS EXPERTOS

PREFACIO

El presente trabajo esta diseñado para servir como una guia pedagógica en el estudio de la teoría y aplicación de los Sistemas Expertos, una de las ramas de mayor auge de la Inteligencia Artificial.

El objetivo es dar a conocer la teoría en la que se sustentan los Sistemas Expertos y las herramientas disponibles para desarrollarlos, para ayudar al lector a desarrollar sistemas más sofisticados en campos de aplicación donde la metodología tradicional de programación es poco flexible.

El trabajo se ha estructurado en nueve capítulos, resumidos de la siguiente manera:

En el capítulo I se efectúa un recorrido del proceso evolutivo que ha sufrido durante los últimos años la Inteligencia Artificial, y principalmente los Sistemas Expertos; así mismo se proporcionan algunas de las definiciones que mejor describen a estos conceptos, proporcionadas por reconocidos expertos.

En el capítulo II se determinan los componentes que conforman a un Sistema Experto, así como sus características y sus interrelaciones entre ellos.

El capítulo III realiza una explicación de las principales

representaciones no formales que pueden implementarse en las bases de conocimiento de un Sistema Experto. Se detallan las ventajas y desventajas de cada uno de ellos.

El capítulo IV nos orienta sobre los lenguajes de programación mas utilizados en el desarrollo de Sistemas Expertos, examinando las características más importantes de cada uno de ellos, para permitirnos conocer las ventajas y desventajas que nos ofrecen en la implementación de este tipo de programas. Así mismo, se hace mención de las principales herramientas comerciales que actualmente nos ofrece el mercado.

El capítulo V nos ofrece la posibilidad de conocer el entorno de computadoras que hacen más eficiente la ejecución de estos sistemas, haciendo énfasis en los estudios que actualmente se realizan con miras a la liberación del Proyecto de Quinta Generación.

El capítulo VI nos da a conocer una metodología de desarrollo universalmente aceptada, para formalizar el diseño de Sistemas Expertos mencionando las semejanzas más significativas con respecto a las formas tradicionales de construcción de software.

El capítulo VII analiza con más detalle algunos de los lenguajes de programación más poderosos en la implementación de Sistemas Expertos en microcomputadoras personales.

El capítulo VIII proporciona un sencillo ejemplo de implementación en herramientas comerciales, mientras que el capítulo IX analiza la tendencia en el estudio e investigación de los Sistemas Expertos.

CAPITULO I

INTRODUCCION

Desde hace algún tiempo ha venido hablándose de uno de los temas hoy por hoy más apasionantes en el campo de la informática: la Inteligencia Artificial. Esta ciencia, objeto de numerosas discusiones entre la comunidad informática debido a la dificultad de aceptar la gran mayoría de científicos la presencia de mecanismos pensantes, gana cada vez más popularidad.

En realidad, si nos apegamos a una definición de inteligencia: "Es la capacidad de comprender hechos y proposiciones, sus

relaciones y razonamientos", podríamos concluir que cualquier manejador de base de datos de tipo relacional es un mecanismo inteligente, ya que almacena información (comprender), acepta proposiciones y con ello representa relaciones, además es capaz de manipular dicha información (razonamiento) para proporcionar un resultado. Sin embargo, la principal diferencia radica en la libertad de decisión que poseen los seres humanos, los sistemas computacionales no son aún capaces de salirse del contexto en el que fueron creados, aunque cabe aclarar que los diccionarios actuales no lo consideran como un requisito de inteligencia. La importancia de construir mecanismos basados en Inteligencia Artificial es obvia, contar con dispositivos que busquen asemejarse a la forma de resolución humana, contando para ello con estructuras proporcionadas por la ingeniería de conocimiento.

En los últimos años hemos visto un gran avance de esta ciencia, incluso algunos científicos han mencionado que la Inteligencia Artificial ha pasado ser de un pequeño aspecto de la ingeniería informática a ser, quizá, la aportación más importante al mundo de la computación. Este avance se basa, en mucho, al gran impacto de una de las ramas de mayor éxito comercial de la Inteligencia Artificial: los Sistemas Expertos, debido a la facilidad -cada vez mayor- de diseñarlos. De ahí el interés por abocarnos al estudio de estos Sistemas Expertos, a partir de la teoría sobre la cual se sustentan hasta el desarrollo de pequeñas aplicaciones.

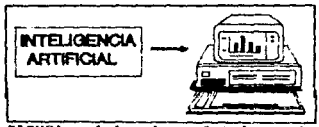


FIGURA 1.1 La Inteligencia Artificial ha dado un nuevo auge a la ingeniería computacional.

I.1 INTELIGENCIA ARTIFICIAL

I.1.1 DEFINICION

Existen muchos autores que proporcionan una definición de la Inteligencia Artificial, algunos de ellas son los siguientes:

[1] "Es la ciencia que trata de la comprensión de la inteligencia y del diseño de máquinas inteligentes, es decir, del estudio y simulación de las actividades intelectuales del hombre, tales como

[1] JUAN PABLO SANCHEZ Y BELTRAN

manipulación, razonamiento, percepción, aprendizaje, creación, etc.".

[2] "La Inteligencia Artificial estudia las máquinas que ejecutan tareas propias del ser humano, en las que se manifiesta la inteligencia; también estudia la modelización del pensamiento".

[3] "La Inteligencia Artificial, es por un lado, un término respetuoso, que define aunque de forma un tanto desatinada, las técnicas de la lógica formal, de los procedimientos y métodos de búsqueda de la representación del conocimiento en programas de computadora".

[4] "La inteligencia Artificial es el estudio de cómo hacer que las computadoras hagan cosas que, en estos momentos hace mejor el hombre".

A mi muy personal punto de vista la definición de Sánchez y Beltrán es la que mejor define este concepto, pues en ella parte de la importancia de la comprensión del proceso de pensamiento del

---

[2] ANA MA.MARTINEZ

[3] KLAUS BAUER

[4] ELAIN RICH

hombre para poder aplicarlo a computadoras que simulen dicho proceso hasta hoy considerada exclusiva del hombre.

La Inteligencia Artificial se divide en varios campos, los más importantes son:

1) La Robótica, es la rama que se encarga del diseño y desarrollo de máquinas capaces de realizar aquellos procesos mecánicos repetitivos y tareas manuales de las cuales es capaz el hombre.

2) Los Sistemas Expertos, son la rama encargada del diseño y desarrollo de sistemas que simulen el proceso intelectual de un experto humano en una materia determinada.

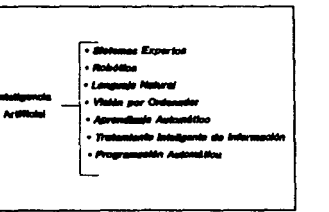


Figura 1.2 La Inteligencia Artificial se divide en varias ramas.

3) El Lenguaje Natural (PLN), estudia el proceso de comunicación entre el hombre y los ordenadores mediante algún tipo de lenguaje natural propio del hombre (el habla, por ejemplo). Es un problema que involucra un estudio sintáctico, semántico y prágmatico de las expresiones del hombre.

4) La Visión por Ordenador, cuyo estudio consiste en la identificación, inspección, localización y verificación de objetos por computadora. Por tanto esta rama está muy relacionada con la de la robótica.

5) El Aprendizaje Automático, es la rama cuyo estudio se enfoca al aprendizaje de programas de computadora de manera automática.

6) El Tratamiento Inteligente de la Información, estudia, como su nombre lo indica, las maneras más inteligentes de manipular la información. En esta rama, apenas se están dando a conocer los primeros lenguajes especializados.

7) La Programación Automática, es la rama que estudia la generación de manera automática de programas que satisfagan una amplia variedad de problemas.

I.1.2. DESARROLLO HISTORICO

Existen muchos acontecimientos en relación con las primeras investigaciones sobre Inteligencia Artificial, una de las primeras fue la aparición de la lógica formal y la psicología cognoscitiva antes de la Segunda Guerra Mundial, que definitivamente fueron un

gran aporte en el desarrollo de modelos de conocimiento. Posteriormente, en el año de 1950, Alan Turing (quien tiempo atrás había construido la llamada "Máquina de Turing", constituyéndose en el primer ordenador que utilizó el concepto de memoria almacenada) ideó una prueba para reconocer comportamientos inteligentes, el cual se le conoce como el Test de Turing.

En este trabajo, Turing proponía la creación de una máquina inteligente, y trataba, al mismo tiempo de adelantarse a las objeciones que pudieran utilizarse contra ella.

En el año de 1955, Alled Newell, J.C. Shaw y Herbert Simon desarrollaron lo que se conoce como el primer lenguaje de la Inteligencia Artificial: el IPL II, pero el que verdaderamente llamó la atención de todos los científicos fue la aparición del lenguaje LISP en el año de 1958, obra de John McCarthy.

En la década de los sesenta aparecen los primeros sistemas que trataban de implementar mecanismos inteligentes y la gran mayoría de ellos fueron orientados al ajedrez, debido principalmente a la innumerable gama de opciones con que este juego contaba. Sin embargo, el hecho más sobresaliente en esta década, fue la aparición del famoso programa ELIZA, escrito en 1964 por Joseph Weinzenbaum, constituyéndose éste, en el primer sistema capaz de reconocer un conjunto del lenguaje oral del hombre. Así también, a finales de esta década se celebra el Primer Congreso Internacional

de Inteligencia Artificial, con la participación de los más connotados científicos.

En los años setenta, las computadoras con grandes memorias eran ya frecuentes y las velocidades de ejecución se habían incrementado notablemente, originando grandes avances en el procesamiento del lenguaje natural, la representación del conocimiento y la resolución de problemas, que pasarían a formar la base para la introducción del primer producto comercial de la Inteligencia Artificial: Los Sistemas Expertos.

En ésta misma década surge el que, en estos tiempos, está considerado como el lenguaje más prometedor de la Inteligencia Artificial: El lenguaje PROLOG, obra de Alain Colmerauer. Este lenguaje en el año de 1980 era el elegido de la comunidad científica europea, mientras que en Norteamérica, LISP tenía el mismo estatus. Sin embargo, esta situación cambió en 1981, a raíz del anuncio hecho por los japoneses de utilizar PROLOG como base de sus computadoras de quinta generación, pero el hecho quizá más importante, era que este lenguaje reunía un conocimiento más profundo del proceso de pensamiento comparado a como lo hacía LISP.

Actualmente, el énfasis en el campo de la Inteligencia Artificial pasa de la investigación a la aplicación.

I.2 SISTEMAS EXPERTOS

I.2.1. DEFINICION

Existen también varias definiciones de los Sistemas Expertos, algunas de ellas son las siguientes:

[5] "Un Sistema Experto o lo que es lo mismo, un Sistema Basado en el Conocimiento, es un conjunto de programas de computadora que son capaces mediante la aplicación de conocimientos de resolver problemas en un área determinada del conocimiento o saber y que ordinariamente requerirían de la inteligencia humana".

[6] "Un Sistema Experto es aquel que puede almacenar el conocimiento de expertos para un campo de especialidad determinada -y muy estrechamente delimitada- y solucionar un problema mediante la deducción lógica. Representan, además, la transición del procesamiento de datos al procesamiento de conocimientos y

[5] Juan Pablo Sánchez y Beltrán

[6] Klaus Bauer

sustituyen al mismo tiempo los algoritmos por mecanismos de inferencia".

[7] "Los Sistemas Expertos son programas computacionales que resuelven problemas específicos en cualquier campo del conocimiento".

[8] "Un Sistema Experto es un programa de computadora que reemplaza a un experto humano".

Esta última definición está basada en el Test de Turing, el cual particulariza para los Sistemas Expertos: "Si la ejecución de un conjunto de programas de computadora puede convencernos de que su comportamiento es el que tendría un experto humano, entonces este conjunto de programas es un verdadero Sistema Experto".

Como lo señalan varios autores, un Sistema Experto tiene varias ventajas en relación a expertos humanos, algunas se muestran en la siguiente tabla:

---

[7] Ana Ma. Martínez Enríquez

[8] Forsyth

CRITERIO	HUMANO	S.E.
Son reproducibles	NO	SI
Vida infinita	NO	SI
Tiempo de resolución	ALTO	BAJO
Eficacia Resolutiva	MEDIA	ALTA
Aproximación	MEDIA	EXACTA

Tabla I.1 Diferencias entre Sistemas Expertos y humanos.

Sin embargo, a diferencia de los humanos, los Sistemas Expertos actuales presentan el inconveniente de no ser todavía capaces de obtener el conocimiento por sí mismos.

En los Sistemas Expertos, el conocimiento debe introducirse previamente, para lo cual es necesario un Ingeniero de Conocimiento que logre plasmar en la aplicación, los conocimientos del experto humano dentro de un modelo preestablecido. Un Sistema Experto debe ser flexible para poder ser modificado o expandido en su conocimiento de una forma sencilla, sin que se afecte al resto del sistema.

A diferencia de un programa tradicional, [1]"los Sistemas Expertos tienen la imperiosa necesidad de estar conformados en unidades elementales independientes, que puedan relacionarse unas con otras y que permitan conocer de maneras más sencilla cuál de ellas ha actuado, cuándo y por qué".

Por lo general, los Sistemas Expertos se utilizan en áreas donde la metodología tradicional no es capaz de resolver el problema de una manera eficaz. Algunos aspectos que diferencian a ambos sistemas son los siguientes:

a) El empleo de un procesamiento simbólico por parte de los Sistemas Expertos frente al de tipo numérico utilizado en los programas tradicionales para resolver problemas. Generalmente los Sistemas Expertos no son utilizados en aplicaciones donde se requiera de un cálculo numérico principalmente.

b) Resolución de tipo heurístico de los Sistemas Expertos contra las resoluciones de tipo combinatorio de los programas tradicionales. Los Sistemas Expertos son utilizados en aplicaciones donde se requiere principalmente de establecer suposiciones (heurísticos) en base a experiencias.

---

[1] DAVID BOLSTON

En los sistemas tradicionales, generalmente todas las situaciones que pueden presentarse están controladas por el programador mediante análisis de tipo combinatorio.

c) En un Sistema Experto la definición del problema es generalmente declarativa mientras que en los otros es procedimental.

d) Las actualizaciones a los Sistemas Expertos son frecuentes, mientras que en los programas tradicionales son raras.

Generalmente los Sistemas Expertos se emplean en áreas del conocimiento donde se efectúa principalmente un procesamiento simbólico de información (medicina, derecho, finanzas, etc.). Su uso tiene significado en ámbitos donde no existe un método definido de solución así como algoritmos que solucionen el problema, y en donde es necesario desarrollar un seguimiento de explicación y justificación de resultados. Además, para considerar su aplicación, el problema planteado debe [16]"requerir conocimiento, juicio y experiencia, principalmente".

En consecuencia, los Sistemas Expertos emplean técnicas de diseño novedosas por las características con que debe contar.

---

[16] JUAN PABLO SANCHEZ Y BELTRAN

I.2.2. DESARROLLO HISTORICO

La mayoría de los expertos en la materia dividen la historia de los Sistemas Expertos en tres etapas.

La primera de ellas abarca hasta el año de 1974, en el que surge la teoría sobre la que están construídos estos sistemas; se desarrollan los primeros lenguajes de programación orientados a esta rama y se diseñan ordenadores capaces de satisfacer los requerimientos que se necesitan.

En la segunda etapa, de 1974 a 1984, los Sistemas Expertos empiezan a tomar un gran auge, de hecho a esta etapa se le conoce en el mundo científico como "La década de los Sistemas Expertos". Durante el desarrollo de esta década empiezan a surgir las primeras grandes aplicaciones, entre las que se cuentan: MYCIN (un Sistema Experto para diagnósticos médicos), XCON (Un Sistema Experto para configuración de computadoras), PROSPECTOR (Un Sistema Experto para estudio de yacimientos de minerales), etc. En esta etapa los japoneses lanzan su proyecto de computadoras de quinta generación en el cual dichas computadoras llevarían como base el lenguaje de programación PROLOG, con lo que el interés fue aún mayor.

La tercera etapa comienza en 1984 y todavía seguimos en ella. Se caracteriza por la gran difusión de lenguajes especializados, así como de herramientas y sobre todo, de Sistemas Vacíos, que han

permitido la implantación de Sistemas Expertos sin necesidad de tener grandes conocimientos en informática.

Se ha establecido dentro de la comunidad informática que la finalización de esta etapa tendrá lugar con la comercialización de computadoras especializadas y de lenguajes paralelos, y posiblemente ello sucederá con la liberación del Proyecto de Quinta Generación de los japoneses.

CAPITULO II

ESTRUCTURA DE LOS SISTEMAS EXPERTOS

II.1 INTRODUCCION

Los Sistemas Expertos estan conformados de cinco módulos que actúan de manera independiente:

- El motor de inferencia, representa la unidad de control del sistema, encargada de proporcionar las soluciones del problema planteado.

- La base de conocimiento, conformada por el dominio de la aplicación a tratar, desarrollada en alguno de los diversos tipos de representación del conocimiento.

- La base de hechos o memoria de trabajo, conformada con los datos invariables que en ese momento se tienen en el sistema (hechos u objetivos de reglas cumplidas).

- Interfaz usuario-sistema, mediante el cual se efectúan las consultas a la base del conocimiento del Sistema Experto.

- Interfaz experto-sistema, mediante el cual se tiene acceso a la modificación de la base de conocimiento de nuestro sistema.

Un diagrama que nos muestra la relación que guardan estas unidades se encuentra en la figura 2.1.

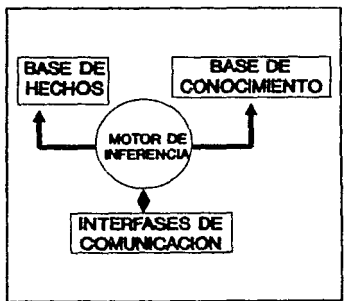


Figura 2.1 Interrelación entre los componentes de un Sistema Experto.

II.2 EL MOTOR DE INFERENCIA

"El motor de inferencia es la parte del Sistema Experto que actúa como control del mismo, es aquel que construye de manera dinámica las soluciones", utiliza la información suministrada por el usuario para encontrar un conjunto de posibles soluciones que satisfagan al problema planteado, auxiliándose para ello de la información existente en la base de conocimiento y la base de hechos del sistema, y generando las soluciones en la misma base de hechos.

El motor de inferencia tiene la función de proporcionar las posibles soluciones al problema planteado. Esto se consigue mediante la ejecución de las unidades de conocimiento (reglas, nodos, objetos, etc.) encontradas en la base y almacenándolas en la base de hechos del sistema. Cada vez que se accesa una unidad de conocimiento a la base de hechos, el motor de inferencia verifica en ésta si ya existe una solución al problema planteado para que la ejecución se detenga o de lo contrario, continúe.

Una conclusión se produce mediante la aplicación de un conjunto de reglas sobre los hechos presentes.

[1] JUAN PABLO SANCHEZ Y BELTRAN

Ejemplo: El objetivo es conocer si LLUVIA se cumple de acuerdo a la regla y hechos siguientes

Regla: SI NUBLADO y FRIO ENTONCES LLUVIA

Hechos conocidos: NUBLADO y FRIO.

Solución:

El mecanismo de inferencia busca en nuestra base de conocimiento si los subobjetivos NUBLADO y FRIO se encuentran presentes. Si es así, se considera a LLUVIA como una conclusión válida y pasa a formar un nuevo hecho, de lo contrario la aseveración es errónea.

Este proceso es muy similar al utilizado por un humano: Es capaz de deducir un hecho siempre y cuando tenga un conocimiento previo válido.

Es importante tener presente el grado de validez de una regla, ya que en el caso del ejemplo anterior, nuestro sistema nos informará que habrá LLUVIA si está NUBLADO y hace FRIO, lo que en realidad no es necesariamente cierto, por lo tanto habrá que elegir cuidadosamente el conocimiento que implantemos en nuestra base de conocimiento.

Cuando una regla se ha aplicado sobre algunos hechos, se "dispara", provocando la inserción de la conclusión respectiva en la memoria de trabajo del Sistema Experto. En el caso del ejemplo anterior, la regla se "disparó" al cumplirse los hechos que la sustentaban y el objetivo LLUVIA pasó a formar parte de la base de

hechos.

Podemos concluir que el funcionamiento del motor de inferencia consiste en seleccionar, comprobar y ejecutar el conjunto de reglas o conocimientos que se emplean dada una entrada.

II.2.1. CARACTERISTICAS

La característica más importante del motor de inferencia es que es una unidad totalmente independiente de la base de conocimiento y de la base de hechos, lo que permite una gran flexibilidad en el sistema, posibilitando la implantación de más conocimientos al sistema o incluso modificar los existentes sin necesidad de alterar el mecanismo de control.

Es importante mencionar que esta característica de independencia ha evolucionado con el transcurso de los años; anteriormente los motores de inferencia diseñados tenían una estructura muy rígida, ya que funcionaban exclusivamente con la base de conocimientos original, generando sistemas poco flexibles y muy costosos. La evolución de estos mecanismos de inferencia ha sido sorprendente ya que en la actualidad se comercializan "caparazones" de Sistemas Expertos[1], donde el usuario únicamente tiene que proporcionar la

[1] Paquetes de software donde el motor de inferencia ya viene implementado y la base de conocimiento es diseñada por el usuario.

base de conocimiento.

Existen otras características también muy importantes, como los mecanismos de búsqueda de soluciones, la elección del conocimiento, el metaconocimiento, la lógica utilizada, la evaluación del conocimiento, etc.

II.2.2. MECANISMO DE BUSQUEDA

Los mecanismos de búsqueda se refieren a la forma en que el motor de inferencia busca dentro de la base de conocimiento las posibles soluciones al problema presentado. Esta búsqueda puede ser ordenada o no ordenada.

Una búsqueda no ordenada es aleatoria o heurística.

Una búsqueda no ordenada es aleatoria cuando nuestro motor de inferencia realiza una búsqueda en toda la base de conocimiento, consiguiendo que se tengan todas las soluciones probables, sin embargo, debemos considerar que la rapidez de este proceso está en relación directa con el tamaño de la base, cuanto más grande es ésta, más lenta es la búsqueda.

La búsqueda no ordenada heurística ocurre cuando el motor de inferencia utiliza metaconocimientos, es decir, estrategias de

solución avanzadas que permiten que el campo de posibles soluciones se reduzca mediante la utilización de hipótesis o heurísticos. Este método mejora el tiempo de ejecución, su éxito depende en mucho del grado de validez de los procedimientos heurísticos planteados.

Los mecanismos de búsqueda más utilizados en los Sistemas Expertos son los ordenados, basados en el encadenamiento del conocimiento, es decir, provocando que la salida de una regla, pase a ser la entrada de la siguiente regla a considerar. Un esquema del seguimiento en la búsqueda de soluciones de un mecanismo de búsqueda ordenada es semejante a la estructura que tiene un diagrama de árbol.

Este tipo de búsqueda permite también, la utilización de procedimientos heuristicos, agilizando aún más el tiempo de ejecución, y posibilitando lo que se conoce como "poda de arbol", es decir, reduciendo el campo de posibles soluciones.

Existen 3 tipos de búsqueda ordenada:

a) El encadenamiento hacia adelante (forward chaining), es conocido también como deductivo o conducido por datos. Este mecanismo requiere de datos de entrada para que el motor de inferencia genere nuevos hechos, hasta encontrar el objetivo principal.

Esta técnica busca en la base de conocimiento, reglas que puedan

ser "disparadas" (ejecutadas) a partir de los hechos ya conocidos, y se verifica en cada paso si se ha llegado al objetivo previsto.

Ejemplo:

Supóngase que en nuestra base de conocimiento tenemos lo siguiente:

Regla 1 Si NUBLADO y FRIO, entonces **LLUEVE**.

Regla 2 Si **NEBLINA**, entonces **NIEVA**.

Regla 3 Si **LLUEVE** y **NIEVA**, entonces **TORMENTA**.

Regla 4 Si **LLUEVE** y **NEBLINA**, entonces **GRANIZO**.

Y en nuestra base de hechos:

NUBLADO.

FRIO.

NEBLINA.

El objetivo es encontrar si el objetivo **GRANIZO** se cumple a partir de esta información.

SOLUCION:

La metodología que sigue el encadenamiento hacia adelante es ejecutar aquellas reglas lo puedan hacer, verificando en cada paso si el objetivo se ha cumplido.

PASO 1) Se encuentra que **Regla 1** y **Regla 2** pueden ser disparadas, con lo que se elige **Regla 1** por ser la primera que se encuentra. Así, se dispara **Regla 1** e introduce **LLUEVE** en la base de conocimiento y el sistema "memoriza" que la regla ha sido aplicada. El objetivo no se ha cumplido.

PASO 2) Se descubre que **Regla 2** y **Regla 4** pueden ser disparadas, con lo que se elige **Regla 2**, por ser la primera que se encuentra. Así, se dispara **Regla 2** e introduce **NIEVA** en la base de conocimiento y el sistema "memoriza" que la **Regla 2** ha sido aplicada. El objetivo aún no se ha cumplido.

PASO 3) Se descubre que **Regla 3** y **Regla 4** pueden ser ejecutadas, eligiendo **Regla 3**, por ser la primera. Se introduce **TORMENTA** en la base de conocimiento y el sistema "memoriza" que la **Regla 3** ha sido aplicada. El objetivo no se ha cumplido.

PASO 4) Se descubre que la **Regla 4** se puede disparar, lo que ocurre y con ello introduce **GRANIZO** en la base de conocimiento, permitiendo el cumplimiento del objetivo buscado.

En el mecanismo de búsqueda de encadenamiento hacia adelante los atributos o datos, que definen a los objetivos, conducen a las soluciones (figura 2.2) y el diagrama de seguimiento muestra a una estructura de "árbol" construida de las "hojas" a la "raíz".

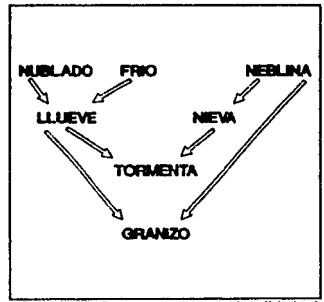


Figura 2.2 Estructura de "árbol formada por un mecanismo de encadenamiento hacia adelante.

Una de las mayores desventajas de este mecanismo es la realización de búsquedas innecesarias en la base de conocimiento. En el caso del ejemplo anterior el paso 3 es inútil,

ya que no tiene ningún sentido la ejecución de una regla que defina a un subobjetivo no necesario (TORMENTA).

b) El encadenamiento hacia atrás, denominado también guiado por objetivos o inductivo, es el mecanismo de busqueda más empleado por los Sistemas Expertos actuales. Consiste en partir de un objetivo o conclusión para verificar si la regla que lo define se cumple o no, probando para ello a cada uno de sus atributos.

Ejemplo:

El usuario desea conocer si el estado GRANIZO se cumple o no, considerando la base de conocimiento anterior.

SOLUCION:

La metodología que se sigue es buscar en la base de conocimiento una regla que defina al objetivo deseado y tratar de verificar a cada uno de los subojetivos.

PASO 1) El motor de inferencia busca si GRANIZO está contenida en la base de hechos (NUBLADO, FRIO, BRILLNA), y descubre que no lo está, por lo que introduce GRANIZO como hipótesis y se intenta verificarla en el siguiente paso.

PASO 2) Se descubre que la Regla 4 describe al objetivo GRANIZO, y se procede a verificar la primera premisa de dicha regla (LLUEVE). El motor de inferencia no encuentra a LLUEVE en la base de hechos, y la acepta como hipótesis.

PASO 3) Se descubre que la Regla 1 describe al subojetivo LLUEVE y se procede a probar la primera premisa

(NUBLADO), la cual está contenida en la base de hechos.

Se verifica la segunda premisa de la Regla 1 (FRIO), descubriéndose que también está contenida en la base de hechos. Con ello se cumplen todas las premisas de la Regla 1, provocando que LLUEVE sea un nuevo hecho.

PASO 4) Se comprueba como continuación del paso 2, si la segunda premisa de la Regla 4 se cumple (NEBLINA), descubriendo que está contenida en la base de hechos.

Con ello se han cumplido todas las premisas de la Regla 4, provocando el "disparo" de GRANIZO. El proceso de encadenamiento termina exitósamente.

El diagrama del seguimiento de soluciones de un sistema de encadenamiento hacia atrás construye un "árbol" desde la "raíz" hasta las "hojas", como se muestra en la figura 2.3.

Este mecanismo es mucho más rápido que el anterior debido a que no se realizan búsquedas innecesarias.

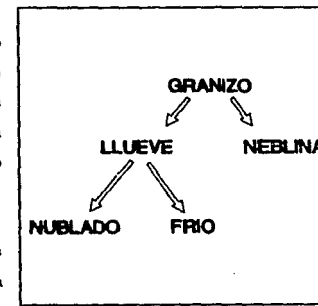


Figura 2.3 Estructura de "árbol" formada por un mecanismo de encadenamiento hacia atrás.

c) **El encadenamiento mixto;** consiste en la aplicación del

encadenamiento hacia atrás auxiliado del encadenamiento hacia adelante para la búsqueda de soluciones.

Ejemplo:

Supongamos que se tiene la Base de Conocimiento siguiente:

Regla 1 presion si $(dv/dt)^1$ y fuerza

Regla 2 presion si fuerza y $(d^1x/dt^2)^2$

Regla 3 $(dv/dt)^1$ si aceleracion

Regla 4 fuerza si aceleracion y masa

Y los hechos:

$(dv/dt)^2$ y masa.

El objetivo es conocer si se cumple "presion".

SOLUCION:

- Se aplica el mecanismo de encadenamiento hacia atrás con el fin de demostrar presión.

- Se aplica la Regla 1, $(dv/dt)_1$ está contenida en la base de hechos y el subobjetivo fuerza se define mediante la Regla 4.

- Se aplica la Regla 4, aceleracion^1 no es conocido por lo que no se cumple, y falla la Regla 4 con lo que la Regla 1 no se cumple.

- Se aplica la Regla 2, pero falla al hacerlo la Regla 4.

La marcha atrás finaliza sin haber conseguido llegar a la solución. Se efectúa entonces, el mecanismo de encadenamiento hacia adelante.

- Se busca una regla que pueda aplicarse y se encuentra que la Regla 3 puede hacerlo, "disparando" al subobjetivo aceleracion[1].

- Se reinicializa el encadenamiento hacia atrás: se cumple la Regla 1 al cumplirse la Regla 4, y se demuestra que presion es una hipótesis válida.

Puede observarse que sin el auxilio del mecanismo de búsqueda hacia adelante, el sistema no habría sido capaz de resolver el problema planteado con la información suministrada.

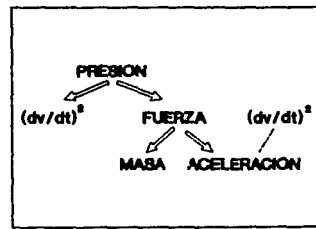


Figura 2.4 Estructura de "árbol" formada por un mecanismo de encadenamiento mixto.

El 'arbol' que se generó es como el que se muestra en la figura 2.4.

Una estructura de 'arbol' como la de este tipo, puede ser analizada en 2 sentidos:

1). En anchura o "breadth first search", en la que se comienza el proceso de búsqueda mediante la expansión del estado inicial. Si ninguno de estos nodos es un estado objetivo, se genera el siguiente nivel de búsqueda. Este nivel se produce tomando uno a uno los nodos anteriores y aplicando sobre ellos operadores específicos. El proceso continúa hasta encontrar el objetivo buscado.

La desventaja de esta técnica radica en el crecimiento exponencial del número de nodos en cada nivel, llegando en ocasiones a tener un número muy grande de posibilidades, que imposibilita su manejo.

2). En profundidad o "depth first search", en la que se selecciona una trayectoria del nodo inicial, haciendo el análisis en niveles cada vez más profundos, hasta que se descubre una solución, o el final del camino. Si no existe una solución, el proceso se reinicializa con otra trayectoria del nodo inicial.

Ejemplo:

Consideremos la base de conocimiento siguiente que da origen a una estructura de árbol similar al de la figura 2.5.

Base de Conocimiento

Si C y I entonces E

R: F y G entonces I

R: B y C entonces A

En una búsqueda en anchura el orden sería:

A-B-C-D-E-F-G

En una búsqueda en profundidad el orden sería:

A-B-D-E-C-F-G

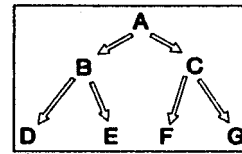


Figura 2.5 Estructura de "árbol" formada por la base de conocimiento dada.

II.2.3. ELECCION DEL CONOCIMIENTO.

El motor de inferencia de un Sistema Experto tiene la función de buscar las soluciones y elegir de entre ellas, la que considere óptima.

Para estudiar la manera en que el Motor de Inferencia elige el conocimiento a aplicar, inspeccionaremos en principio, como funciona el mecanismo de control de un lenguaje procedimental.

En un lenguaje procedimental, tal como BASIC, COBOL, FORTRAN, etc., la elección de la solución se efectúa bajo un mecanismo de tipo imperativo, es decir, el programador indica el orden de ejecución de las instrucciones en el programa. En este caso el programador tiene un control completo sobre el sistema.

Como el hombre no podría trabajar bajo este mecanismo de control para elegir conocimientos, debido a que su modo de actuar es según los datos que reciba, se han desarrollado en los Sistemas Expertos, mecanismos que se acerquen más a este tipo de comportamiento. El lenguaje PROLOG en específico, tiene un motor de inferencia, llamada máquina PROLOG, capaz de realizar la elección de soluciones de manera muy similar a como la efectúa un humano, este lenguaje utiliza el proceso de unificación en la elección del conocimiento.

[3]"La unificación es el proceso por el cual se consigue una uniformidad de la estructura uniendo valores a las variables, es decir, instanciándolas". Haciendo una analogía con las funciones, puede decirse que la unificación es el proceso mediante el cual se consigue el paso de parámetros a una función.

Ejemplo:

Supongamos que nuestra base de hechos contiene la información siguiente:

```
programador(carlos).
programador(maria).
```

Y tenemos la petición de usuario:

```
Goal> programador(Quien)
```

Solucion:

---

[3] KLAUS BAUER

Mediante el proceso de unificación de PROLOG la solución proporcionada es: Quien=carlos Quien=maria

PROLOG unifica la variable Quien a los valores 'carlos' y 'maria'. El funcionamiento de la máquina PROLOG es como lo señala la siguiente figura proporcionada por Juan Pablo Sánchez y Beltrán.

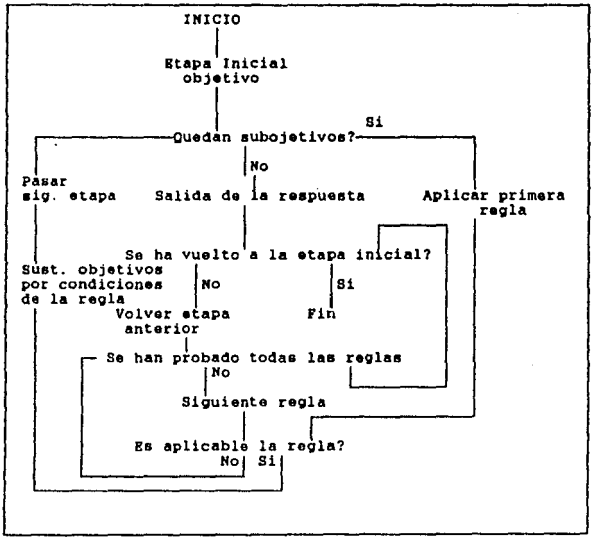


Figura 2.6 Máquina PROLOG

II.2.4. METACONOCIMIENTO.

[4] "El metaconocimiento, es un conocimiento que sirve como auxiliar al motor de inferencia en la elección de soluciones, permitiendo la selección y aplicación de una serie de conocimientos específicos".

El metaconocimiento se puede presentar de diversas formas:

- El metaconocimiento implícito, se localiza en el mismo motor de inferencia del lenguaje. Ejemplo de esto, es la búsqueda en profundidad y la vuelta atrás en PROLOG o GURU, cuyos motores de inferencia las contemplan de manera automática.

- Proporcionado por el programador, para realizar un control de tipo estratégico que auxilie en la búsqueda de soluciones. Ejemplos en PROLOG, son los predicados CUT y FAIL, los cuales impiden la vuelta atrás (CUT) o la permiten (FAIL) como parte del control de un programa.

- Proporcionado por el usuario, durante la comunicación

[4] Juan Pablo Sánchez y Beltrán

interactiva con el Sistema Experto.

Los metaconocimientos permiten búsquedas más eficientes, sobre todo en el sentido de la rapidez de respuesta del Sistema Experto.

Ejemplo:

Consideremos el siguiente programa en PROLOG:

```
DOMAINS
  persona = symbol
PREDICATES
  gripa (persona)
  enfermo (persona)
  tose (persona)
CLAUSES
  gripa (X) if
    enfermo (X) and tose(X) and !.
  enfermo (pedro).
  enfermo (juan).
  tose (pedro).
  tose (juan).

Goal: gripa (Quien)
      Quien=pedro
      1 solución
```

Observe el uso del predicado corte (!), un metaconocimiento proporcionado por el programador, que impide la vuelta atrás del mecanismo de inferencia y con ello permite que sólo se genere una solución.

II.2.5. LOGICA.

Existen muchos tipos de lógicas utilizadas en los Sistemas Expertos, y dos que llaman poderosamente la atención debido a su

capacidad y legibilidad son: la lógica proposicional y la lógica de cálculo de predicados.

[5]"La lógica proposicional trata de la determinación de la verdad o falsedad de las proposiciones". Una proposición es una sentencia que puede ser falsa o verdadera.

Virtualmente, la mayoría de los lenguajes de programación utilizan la lógica proposicional como base para el control de un programa, utilizando para ello los distintos operadores con que cuenta: Y, O, NO, IMPLICA y EQUIVALENTE.

Ejemplo: IF ventas > 100 AND mes >= 10 THEN ok=.t.

Una de las lógicas más potentes en la actualidad es la lógica de cálculo de predicados, también llamada lógica de primer orden, y es la que implementa el lenguaje PROLOG.

La base del cálculo de predicados es el predicado mismo, el cual es, esencialmente, una función que devuelve el valor de verdadero o falso dependiendo de su argumento. Un aspecto importante para considerar el cálculo de predicados sobre la lógica proposicional es la facilidad para implementar relaciones entre objetos.

---

[5] HERBERT SCHILDT

Ejemplo:

La representación de una buena escuela y otra que no lo es, en lógica de cálculo de predicados es:

buena_escuela("Esep aragón") ----------> true

buena_escuela("Universidad x") ----------> false

En lógica proposicional se representa:

buena_escuela = esep_aragon

mala_escuela = universidad_x

Observe que en la lógica proposicional, debe utilizarse una sentencia nueva para cada caso, lo que no es necesario en el cálculo de predicados. Esta característica representa una gran ventaja del cálculo de predicados, ya que permite la utilización de menos código en el desarrollo de sistemas.

II.2.6. EVALUACION DEL CONOCIMIENTO.

Según la forma en que efectúan la evaluación del conocimiento, un motor de inferencia puede ser determinístico o probabilístico.

En el motor de inferencia determinístico todas y cada una de las

reglas son tomadas como totalmente ciertas.

Ejemplo:

Una regla determinística tiene un 100% de probabilidad:

SI escribe rapido Y usa_micro ENTONCES persona[capturista]

Esta señala que una persona que escribe rápido y usa micro es definitivamente un capturista.

En el motor de inferencia probabilística, las reglas o hipótesis por confirmar tienen un grado de probabilidad en función de los hechos que la sostienen. La probabilidad de una regla puede calcularse aplicando el criterio de Bayes, donde es necesario conocer previamente las probabilidades de cada una de las conclusiones respecto de los hechos o evidencias.

Ejemplo:

Una regla probabilística debe especificar el grado de probabilidad:

```
SI persona[pasante) ENTONCES (persona[desempleada)]
                    PROBABILIDAD 60
```

Esta regla sugiere que si una persona es pasante, la probabilidad de estar desempleado es de 60%.

En la implementación de un motor de inferencia probabilística es necesario desarrollar algoritmos para la evaluación del conocimiento.

Los pasos a desarrollar para la determinación de la probabilidad de una regla son los siguientes:

a) Se determina la probabilidad de los hechos que intervienen en las condiciones de cada una de las reglas. Por ejemplo:

```
presidente(arturo, 0.4).
```

b) Se obtiene la probabilidad de la regla a considerar, en la que puede aplicarse la regla de Bayes (llamada tambien el teorema de probabilidad de causas):

$$P(Ci\backslash H) = \frac{P(H\backslash Ci) * P(Ci)}{\sum_{j=1}^{n} P(H\backslash Cj) * P(Cj)}$$

Donde:

P = Probabilidad
Ci= Creencia especifica
H = Hechos o evidencias
n = Numero de conclusiones posibles
P (Ci\H) = Probabilidad de que Ci ocurra dada la evidencia H
P (H\Ci) = Probabilidad de que este presente el evento H dada la creencia Ci

Ejemplo:

Se va a elegir un presidente de un pais I. Para los distintos candidatos, se tienen las siguientes probabilidades:

P(arturo)= 0.4
P(aracelli)= 0.1
P(maria)= 0.5
P(Incremento/arturo)= 0.2
P(Incremento/aracelli)= 0.8
P(Incremento/maria)= 0.3
P(maria/Incremento)= ??

Aplicando la regla de Bayes:

$$P(maria \backslash Incremento) = \frac{0.3*0.5}{(0.3*0.5)+(0.2*0.4)+(0.8*0.1)}$$

P(maria/Incremento)= 0.4839

La regla de Bayes es implementada por el programador en lenguajes como PROLOG, sin embargo, en una gran variedad de herramientas actuales se aplica de forma automática. Una implementación sencilla es la siguiente:

```
DOMAINS   /* Se define el dominio de las variables */
          /* utilizadas */
  persona=symbol
  numero=real

PREDICATES /* Se definen los predicados a utilizar */
           /* en la seccion de cláusulas, así como */
  presidente(persona,numero)   /* los argumentos  */
  incremento_si_presidente(persona,numero) /* que */
  presidente_si_incremento(persona) /* conforman  */
           /* a cada uno de ellos.               */
```

```
CLAUSES     /* Sección de cláusulas                */

  presidente(arturo,0.4). /* Se definen las probabilidades */
  presidente(araceli,0.1). /* para ser presidente por c/u  */
  presidente(maria,0.5).  /* de los candidatos             */

  incremento_si_presidente(arturo,0.2). /* Se definen las  */
  incremento_si_presidente(araceli,0.8).  /* probabilidades*/
  incremento_si_presidente(maria,0.3).  /* del evento incremento si está presente la creencia de ser presidente
                                           de los distintos candidatos */

   presidente_si_incremento(X) if /* Se define la       */
    presidente(X,Pb1), /* probabilidad  de que un candidato */
    incremento_si_presidente(X,Pb2), /* ocupe la presidencia*/
    Numerador=Pb1*Pb2,   /* si está presente el evento */
    Denominador=(0.4*0.2)+(0.1*0.8)+(0.5*0.3), /* incremento*/
    Prob=Numerador/Denominador,
    write("La probabilidad es ",Prob).
```

El programa permite al usuario preguntar sobre la probabilidad de que una persona en especial sea presidente, si consideramos al evento incremento como presente. La consulta puede efectuarse de la siguiente manera:

```
          > GOAL: presidente_si_incremento(maria)

    La probabilidad es ) 4839
```

Los sistemas expertos actuales emplean la regla de Bayes para verificar el cumplimiento de objetivos. Generalmente si la probabilidad de la regla es mayor a 50 se "dispara", de lo contrario el objetivo buscado no se cumple.

II.3 LA BASE DE CONOCIMIENTO

Es la parte del Sistema Experto que contiene el conocimiento sobre el dominio a tratar en un formato preestablecido.

Este conocimiento tiene que representarse en forma concreta en la base de conocimiento, para ello es necesario que se consideren los siguientes atributos:

- **Sencillez**, para permitir un manejo fácil de la información contenida.

- **Independiente**, para que sus posibles modificaciones no afecten la estructura del Sistema Experto, tanto en la misma base de conocimiento como en su mecanismo de inferencia.

- **Transparente**, para implementar un sistema de justificación y explicación donde se oriente al usuario sobre las resoluciones a las que se han llegado.

- **Relacional**, para establecer relaciones entre los conocimientos implementados.

Un factor importante en el diseño de la base de conocimiento es su capacidad, y ésta se mide de acuerdo al tipo de representación implantado. En redes semánticas, la capacidad se mide en base al número de nodos, y en reglas de producción mediante el número de reglas. En la actualidad, los Sistemas Expertos operacionales

llegan a tener un promedio de 200 a 4000 elementos de conocimiento. Como dato curioso, cabe decir, que expertos en psicología cognoscitiva han concluído que la base de conocimiento de un humano llega a ser hasta de 2,000 millones de unidades cognoscitivas.

II.4 LA BASE DE HECHOS.

[6]"La base de hechos es un conjunto de información que forma el universo del Sistema Experto, y con base a ellos, y mediante la base de conocimiento, el Sistema Experto llega a la solución".

Los hechos son afirmaciones que se hacen de algún objeto en especial y que siempre van a estar presentes, son de caracter invariable.

La gran mayoría de expertos, entre ellos Forsyth, denominan a la base de hechos como memoria de trabajo.

La memoria de trabajo es una región donde son almacenados los hechos de caracter invariable proporcionados por el ingeniero de conocimiento, así como las conclusiones que se "dispararon" a partir del cumplimiento de las reglas que las definían.

---

[6] JUAN PABLO SANCHEZ Y BELTRAN

Es posible visualizar a la base de hechos como una estructura de pila, donde los objetivos de las reglas que se van ejecutando, se van almacenando, y al final son leídas para observar la línea de razonamiento que ha seguido el Sistema Experto.

II.5. LOS MODULOS DE COMUNICACION.

Un módulo de comunicación, es la interfase entre el Sistema Experto y la persona que lo utilizará.

Un módulo de comunicación de un Sistema Experto debe buscar asemejarse a estructuras cercanas al lenguaje natural para permitir un mejor entendimiento por parte del usuario: debe ser simple, no complicado y rápido.

Para poder ser amigable, un módulo de comunicación de Sistema Experto debe de aprovechar las facilidades que otorgan diversas herramientas tales como las pantallas de alta resolución, los menús, las ventanas, los íconos, los colores, animaciones, sistemas de ayuda, etc.

II.5.1. MODULO DEL EXPERTO.

En el módulo del experto o de trabajo, el experto suministra la información al sistema mediante la ayuda del ingeniero de conocimiento. El conocimiento puede ser implementado en el proceso de construcción del sistema, o bien, durante la existencia del mismo, para modificarlos e incrementarlos.

Las modificaciones se realizan en algunas ocasiones, para corregir fallas en el diseño de la base de conocimiento, como pueden ser la detección de redundancias e inconsistencias.

II.5.2. MODULO DEL USUARIO

El módulo del usuario o de consulta, es diseñado para la utilización del usuario del sistema.

En este módulo el usuario suministra la información de entrada, para que el Sistema Experto informe de los resultados obtenidos, así como de sus explicaciones y justificaciones. Es necesario, entonces, diseñarlos de la manera más amigable al usuario utilizando técnicas de presentación, tales como los menús pull down, iconos, etc.

II.6. APRENDIZAJE.

En los Sistemas Expertos el aprendizaje de conocimientos puede realizarse de 2 formas: por transmisión y por inducción.

El aprendizaje por transmisión ocurre cuando el experto suministra o transmite los conocimientos al sistema. Esta transmisión de conocimiento, en PROLOG por ejemplo, se da en forma de reglas que deben ser claras para una óptima utilización de los mismos por parte del Sistema Experto.

El aprendizaje por inducción ocurre cuando el ingeniero de conocimiento se ve ante la imposibilidad de representar el conocimiento del experto y acude entonces, a la utilización de ejemplos específicos que permiten utilizar el método de inducción en la solución de otros problemas de tipo similar.

CAPITULO III

REPRESENTACION DEL CONOCIMIENTO

III.1. INTRODUCCION.

Para el procesamiento de la información contenida en la base de conocimiento del Sistema Experto, es necesario crear representaciones del conocimiento que estructuren, clasifiquen y jerarquizen dicho conocimiento de la mejor forma posible.

Una representación del conocimiento consta de una estructura que describe los componentes del conocimiento, así como un módulo interpretativo del mismo.

En general, un sistema de representación del conocimiento debe

ser:

- Transparente, para distinguir con facilidad los conocimientos almacenados.

- Natural, para representar el conocimiento en su forma original, es decir, sin interpretaciones que dañen la flexibidad del sistema.

- Eficiente, para agilizar los mecanismos de búsqueda de conocimientos.

- Modular, para estructurar conjuntos de conocimientos relacionales como unidades que faciliten la transparencia y eficiencia del sistema.

- Simbólico, para permitir una gran facilidad de descripción.

La psicología cognoscitiva ha creado esquemas de representación bastante acertados, que han permitido que la Ingeniería de Conocimiento los implemente en Sistemas Expertos. Entre los más importantes tenemos: la representación basada en lógica formal[1]

[1] Este tipo de representación tiene un sintema de razonamiento monotónico, que significa "moverse en una dirección únicamente", y que permite que el número de hechos verdaderos siga siempre una línea ascendente.

La principal desventaja es la imposibilidad de desarrollar creencias tentativas o probables, que constituyen una parte importante del comportamiento humano.

como el cálculo de predicados, y las no formales como las reglas de producción (rules), los marcos (frames), las redes semánticas (nets) y los objetos (objects). En este trabajo presentaré los tipos de representaciones no formales ya que guardan un comportamiento más similar al humano, además de ser los más utilizados.

III.2. REGLAS DE PRODUCCION.

La representación del conocimiento en forma de reglas de producción es la más utilizada actualmente y fue propuesta por Post en 1943. Consta de 3 secciones:

a) Una region de memoria, para almacenar el estado actual del universo en cuestión (reglas disparadas, valor de variables, etc.).

b) Un conjuto de reglas, de la forma:

(<nombre producción>
{ <elemento condicional> }
....
{ <elemento condicional> }
------>
{ <acción> }

donde:

<nombre producción> , es el nombre de la regla en cuestión.

<elemento condicional>, es una estructura del tipo:

- <identificador de atributo> <valor>

donde:

<identificador de atributo>, es el atributo del elemento a comparar.

<valor>, es el valor que debe tener el identificador de atributo para que la condición sea válida.

<accion>, es lo que deberá realizarse si el conjunto de condiciones es válida.

c) Un intérprete, para la ejecución de reglas.

El intérprete del sistema de representación del conocimiento ejecuta el conjunto de reglas cuyas condicionantes han sido satisfechas, lo que provoca la adición de los nuevos hechos en la región de memoria.

```
RULE MANZANA
  forma=redonda AND
  color=rojo OR
  color=amarillo
THEN fruta=manzana;
```

Figura 3.1 Ejemplo de una regla en VP Expert.

La estrategia más utilizada para la selección de reglas en la mayoría de lenguajes para Sistemas Expertos es la siguiente:

a) No tomar en cuenta las reglas que previamente hayan sido ejecutadas.

b) Seleccionar aquella regla cuya mayoría de sus elementos condicionantes hayan sido recientemente aprobados.

c) Seleccionar cualquier regla.

El lenguaje PROLOG permite la implementación de este tipo de representación de manera directa; así también efectúa una restricción en su uso, basándose en el criterio de claúsulas de Horn, la cual afirma que únicamente debe de existir una conclusión por regla, y que la misma no puede aparecer negada.

Ejemplo:

Según el criterio de claúsulas de Horn la siguiente base de conocimiento es errónea sintácticamente en el lenguaje de programación PROLOG:

```
/* Regla 1 */
suerte(X) and nata(X) if
            carnivoro(X), felino(X).

/* Regla 2 */
not carnivoro (X) if
            come_hierba (X).
```

La regla 1 es errónea debido a que existen dos conclusiones en ella, mientras que en la regla 2 la conclusión esta negada

Existe una técnica denominada de metarreglas, la cual es muy utilizada y tiene su base en el criterio de claúsulas de Horn.

Las metarreglas son conocimientos previos que se tienen de las reglas que componen a nuestra base de conocimiento y que permiten la "poda del árbol" en la búsqueda de soluciones. Pueden ser de 2

tipos:

- **Metarreglas ciegas**, son aquellas que contienen el conocimiento sobre la forma física que guardan las reglas. Ejemplo de ello son aquellas metarreglas que seleccionan la regla más corta, o la primera, o la más reciente, en base a los diferentes tipos de entrada que se tengan.

- **Metarreglas inteligentes**, son las más utilizadas debido a que reunen un conocimiento más profundo sobre las reglas a aplicar; se basan en el contenido de las oraciones, es decir, en su significado semántico. Ejemplo de ello, es aquella selección en la que dada una entrada en particular se delimitan un conjunto de reglas en la que se encontrarán las posibles soluciones, basándose en algunas supocisiones, proporcionadas en la mayoría de los casos por experiencias. Dentro de las metarreglas inteligentes existen dos ampliamente utilizadas: la técnica de escalada de la colina y la de menor coste.

TECNICA DE ESCALADA DE LA COLINA

Consiste en el empleo de heurísticos que intenten minimizar el número de conexiones dentro del diagrama de árbol de solución. Para lograr esto, programador debe hacer que el sistema seleccione preferentemente el nodo que requiere de un esfuerzo mayor sobre aquel que requiere de uno menor, basándose en la tesis (válida o

no) de que [1]"la solución estará más cerca conforme mayor sea el esfuerzo que realizemos". La mayoría de las veces este tipo de hipótesis minimiza el número de nodos que nos llevan a la solución.

Ejemplo:

Dado el diagrama de árbol de la figura 3.2 que nos muestra las distintas distancias entre varias ciudades emplearemos la técnica de escalada de la colina para encontrar una trayectoria correcta entre dos ciudades mediante el siguiente heurístico: "conforme más kilometraje avancemos probablemente estemos más cerca de la solución".

Supongamos que el punto inicial es México, y la meta deseada es Tuxtla, entonces la trayectoria que se generaría sería la siguiente:

México-Guadalajara-Tuxtla:

2900 Kilómetros

Esto es debido a que la ciudad con una trayectoria más lejana a partir de México es Guadalajara, que tiene un enlace directo con Tuxtla.

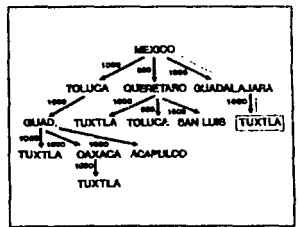


Figura 3.2 Diagrama de árbol formado por la conexión de diversas ciudades.

Observe que este heurístico tiene como objetivo minimizar el número de nodos recorridos en el diagrama de árbol, aunque en el caso de este problema, quizá nos proporcione a la vez una solución con mucho kilometraje recorrido.

TECNICA DE MENOR COSTE

[1] PATRICK HENRY WINSTON

Esta técnica funciona de manera contraria al método de escalando colinas y consiste en obtener la solución de un problema generando aquellos nodos más factibles, es decir los que involucren el menor esfuerzo (costo) posible. La mayoría de las veces, esta técnica genera más nodos en su ruta a la solución que el método anterior, aunque con la ventaja de un menor esfuerzo.

Ejemplo:

Suponganos el mismo árbol del ejemplo anterior, pero ahora determinemos una solución empleando el método de menor costo.

El heurístico que podría servirnos es: "Tomemos aquel vuelo que involucre la distancia más pequeña, ya que probablemente nos de la solución con menor kilometraje de todas".

La trayectoria que obtendríamos como resultado es la siguiente:

México-Queretaro-Puebla: 2600 Km.

Por lo general, la técnica de menor coste maximiza el número de nodos que llevan a la solución aunque nos proporciona principalmente una cuantitativamente mejor en comparación al de escalando colinas, como en este caso, en donde el resultado es el óptimo.

Aunque en este ejemplo el método de menor coste resolvió mejor el problema, no quiere decir que sea superior al de escalando colinas; todo depende de la aplicación a utilizar y de la forma en que se quiere la solución principalmente: más conexiones y por ello

quizá más rápida, escalando colinas; o más precisa pero tal vez más lenta debido a la generación de un mayor número de nodos, menor coste.

Como puede observarse, el empleo de metarreglas la efectúa el ingeniero de conocimiento, mediante la utilización de hipótesis bien fundamentadas.

De lo anterior, podemos concluir que las ventajas de las reglas de producción son su carácter declarativo, su sencillez y su capacidad para permitir la implementación de conocimientos en su forma más primitiva, así como su factibilidad para la utilización de metaconocimientos.

Entre las desventajas podemos situar el efecto de una lenta búsqueda, ocasionado por un número de reglas considerablemente grande y a la gran facilidad, si no existe el debido cuidado, de dar origen a contradicciones y redundancias. Para resolver este problema, es necesario considerar heurísticos de solución que auxilien en evitar la búsqueda exhaustiva. Para evitar el problema de las contradicciones e inconsistencias, una gran mayoría de lenguajes y herramientas actuales, implementan mecanismos para la detección de errores.

III.3 REDES SEMANTICAS.

Las redes semánticas (semantics nets) o también llamadas redes asociativas fueron propuestas por Quillian Collins en 1968 y son un método de representación del conocimiento sobre las relaciones que guardan los elementos de un dominio. Un ejemplo se muestra en la figura 3.3.

Como puede observarse, los elementos del dominio están enmarcados gráficamente en rectángulos y las relaciones entre ellos por vectores.

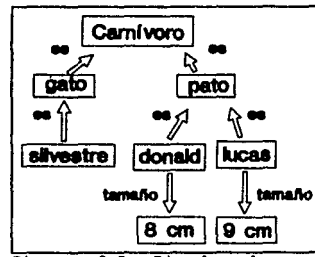


Figura 3.3 Ejemplo de una representación en base a redes semánticas.

Una característica de este tipo de representación es la herencia. Esta especifica que cualquier afirmación verdadera de un elemento del dominio en particular, debe ser también cierta para un ejemplo de dicha clase. Si el siguiente hecho es cierto:

lucas es un pato.

Esta afirmación provoca que lucas tome los atributos de todos los patos, en este caso, carnívoros, por lo que es válida la sentencia:

lucas es un carnívoro.

Esta característica de herencia se representa como:

```
SI relación(x,y) Y relación(y,z) ENTONCES relación(x,z).
```

La cual establece que dada una relación entre "x" y "y", donde "x" es de clase "y", y además existe una relación entre "y" y "z", donde "y" es de clase "z", entonces también existe una relación de "x" y "z", donde "x" es de clase "x". Para el ejemplo anterior, es válida la siguiente expresión:

```
SI es(lucas,pato) Y es(pato,carnivoro) ENTONCES es(lucas,carnivoro)
```

De la misma forma, en una red semántica es posible dar origen a relaciones no mencionadas explícitamente. Un ejemplo es:

```
SI tamaño(an1,t1) Y tamaño(an2,t2) Y (t1 > t2) ENTONCES mayor(an1,an2).
```

La cual establece que dados 2 animales "an1" y "an2", así como sus respectivos tamaños "t1" y "t2", si se conoce que "t1" es mayor que "t2", entonces podemos concluir que el animal "an1" es de mayor proporción que el animal "an2".

El proceso de inferencia de una representación del conocimiento en base a redes semánticas es directo, ya que las asociaciones de conocimientos pueden realizarse haciendo un seguimiento de los

diversos enlaces de la red.

Este tipo de representación del conocimiento es utilizada en Sistemas Expertos para plasmar bases de conocimiento basadas en la lógica proposicional. Sin embargo, sistemas mejor diseñados permiten la implementación de este tipo de representación como un modelo auxiliar para sistemas con reglas de producción principalmente, donde todas y cada una de las situaciones que se pueden presentar están bien definidas en la base de conocimiento. Un ejemplo es la siguiente regla:

```
IF tamaño(X,T1) and tamaño(Y,T2) and T1 > T2
THEN mayor(X,Y)
FC=100
```

La cita anterior. "Si el tamaño de un animal X es T1, y de un animal Y es T2, y además se conoce que T1 es mayor que T2, entonces definitivamente es cierto que el animal X es de mayor proporción que el Y".

En este ejemplo se puede observar la utilización de factores de certidumbre (FC) en Sistemas Expertos, que permiten el procesamiento de conocimientos vagos o probables. En la mayoría de las herramientas actuales los valores posibles para el FC oscilan entre 0 y 100.

Podemos concluir que este tipo de representación tiene su principal ventaja en la facilidad para implementar relaciones entre los conocimientos. Sin embargo, el diseño de una representación del

tipo redes semánticas -en su más puro concepto-, para Sistemas Expertos, es muy difícil debido a la poca flexibilidad que otorgan cuando se trata con estructuras complejas que llegan a formarse cuando se tiene una base de conocimiento considerablemente grande. La solución a este conflicto es como se mencionó anteriormente, diseñar este tipo de representación en combinación con otras, como por ejemplo, las reglas de producción u objetos estructurados.

III.4 OBJETOS ESTRUCTURADOS

III.4.1. MARCOS

Los marcos son objetos estructurados que también son conocidos como "frames" y fueron introducidos por Minsky, en 1975.

[3] "Son una estructura para organizar el conocimiento con énfasis en el conocimiento por omisión".

[4] "Añadido a cada frame hay varios tipos de información, donde parte de ella, hace referencia a como utilizar el frame; otra se

[3] DAVID W. ROLSTON

[4] MARVIN MINSKY

refiere a lo que uno puede esperar que suceda, y otra indica que hacer cuando esta información no es confirmada".

Un marco es una división de un elemento en sus componentes, los cuales son almacenadas en ranuras denominadas "slots", que a su vez pueden estar divididos en "facets" para efectuar una estructuración más precisa. Un ejemplo es el siguiente:

Frame: Vehículos aéreos

Tipo	Omisión: Avión Rango: (Avión,Jet, Helicóptero)
Asientos	Omisión: 40 Rango: (1..80)
Clase	Omisión: Part. Rango: (Part., Pub.)
Aceite	Oil Turbo
Longitud	Omisión: 20 m Rango: (5m..100m)
Control	Omisión: Manual Rango: (Manual,Automat, Remoto)

(A)

Antes de su utilización, un frame es una armazón pre-estructurada de datos. Es en el procesamiento cuando toman valores

los slots que lo conforman.

Estos valores pueden ser heredables, lo que permite que no tengán que redefinirse elementos de conocimiento de tipo general en niveles inferiores, sino unicamente los especificos del objeto en cuestión. La siguiente figura es un marco que ha sido creado de acuerdo con los atributos del marco anterior.

Frame: Vehiculos aéreos
Especificación: Helicóptero

Tipo	Helicóptero
Asientos	4
Clase	Público
Aceite	Oil Turbo
Longitud	10 m
Control	Manual

(B)

Con ello, podemos deducir que un marco puede guardar dos estados:

1) Modelo, prototipo, marco general o vacío, el cual contiene únicamente la estructura del frame y los valores que tendrá por defecto. En el ejemplo anterior, el marco de más alta jerarquía (A) tiene este estado.

2) Particularizado, instanciado o lleno, el cual contiene los slots con valores específicos, además de los valores que le fueron proporcionados por herencia. En el ejemplo anterior, el marco de menor jerarquía (B) tiene este estado.

Como puede observarse, una de las mayores ventajas de los marcos son la capacidad de la opción por omisión que permite un ahorro en el tiempo de búsqueda, ya que toda la información requerida está contenida en el mismo frame.

Dentro de las principales desventajas que se encuentran en este tipo de representación está la dificultad para especificar los valores por omisión de una clase -no todos, observamos de igual forma un objeto-, y sobre todo, del considerable requerimiento de memoria de que se hace uso al tener toda la información disponible en el frame, por lo que su uso es más factible en sistemas de cómputo con amplios recursos.

Los marcos son una representación del conocimiento cuyo uso está creciendo en el desarrollo de Sistemas Expertos, debido a su excelente legibilidad.

III.4.2. OBJETOS

La representación en base a objetos está teniendo un gran auge

en el desarrollo de Sistemas Expertos debido a la sencilla estructura de información que proporcionan y a la factibilidad para implementarla en computadoras personales. Una definición es la siguiente:

[5] "Un objeto es una entidad con un conjunto de atributos y características que describen su naturaleza y funcionalidad".

Los objetos son instancias o ejemplos de las clases a la que pertenecen, y se comunican entre ellos a través de mensajes. Un ejemplo es el siguiente:

Objeto	Clase
Helicoptero	Vehículos aéreos
Jet	Vehículos aéreos
Avion	Vehículos aéreos

El objeto a su vez, está conformado de una estructura de datos y del conjunto de rutinas que las manipulan. Del ejemplo anterior, puede ser válida, la siguiente conformación:

Objeto: Avión

Propiedades:

```
no_llantas : Entero;
motores    : Arreglo de cadena;
aceite     : cadena;
```

[5] LUIS JOYANES AGUILAR

Métodos

```
Procedimiento Aterrizaje ( Argumento: Aeropuerto);
Función Revisa_status ( Argumento: motor): Logico;
```

Como puede observarse, la estructura de datos (no_llantas, motores, aceite) son denominadas las propiedades del objeto, mientras que las rutinas son los métodos que manipulan a dichas propiedades. El paso de mensajes, es la forma en que pueden comunicarse distintos objetos, y ello se realiza mediante la ejecución y comparación de los métodos.

La principal característica de una representación de conocimiento mediante objetos es la herencia.

La herencia es un propiedad que permite la construcción de nuevas subclases de objetos mediante la existencia de clases ya establecidas.

La principal ventaja de una representación en base a objetos es la modularidad que se forma en nuestra base de conocimiento debido a la estructuración como una unidad conjunta de los datos y las rutinas que las manipulan.

Esta representación del conocimiento es muy utilizada en la programación orientada a objetos, la cual puede desarrollarse mediante lenguajes como Smalltalk, C++, Turbo Pascal, Clipper, Actor, etc.

III.4.3. GUIONES

[6]"Los guiones son una especialización del concepto general de un marco, es una estructura que se usa para guardar prototipos de secuencias de sucesos". Constan de cinco componentes:

- Condiciones de entrada, son los valores que deben existir para que la ejecución del guión sea probable.

- Resultados, son los valores que serán válidos después de la ejecución del guión.

- Actores, son los agentes que realizan el guión con el auxilio de las utilerías existentes.

- Utilerías, son los objetos que sirven a los actores para la realización del guión.

- Escenas, son las rutinas que se ejecutarán en el guión.

Ejemplo:

Condiciones de entrada:

Pasajeros en el aeropuerto de Madrid
Destino de pasajeros: México, D.F.

---

[6] DAVID W. ROLSTON

Resultados: Pasajeros en México, D.F.

Actores: Pasajeros, Pilotos

Utilerias: Avión, Altavoz
Sala de espera de aeropuerto de Madrid,
Pasillos de aeropuerto de Madrid,
Aeropuerto de México, D.F.

Escena 1: Abordaje

Pasajeros son llamados en el altavoz
Pasajeros se dirigen a. pasillo
Pasajeros abordan avión

Escena 2: Vuelo

Piloto ejecuta despegue de avión

Escena 3: Llegada

Aterrizaje de avión en Mexico, D.F.
Pasajeros bajan del avion
Pasajeros en México, D.F.

La representación del conocimiento en base a guiones es muy similar a la representación en base a objetos, ya que se definen los datos y los procesos que los manipulan. Sin embargo, es menos utilizada debido a los mayores requerimientos de memoria que solicita, debido a una descripción más detallada de los elementos de conocimiento.

CAPITULO IV

LENGUAJES Y HERRAMIENTAS PARA LOS SISTEMAS EXPERTOS

IV.1 INTRODUCCION

En el presente capítulo analizaremos algunas características de las herramientas y lenguajes de programación más comunes para desarrollar Sistemas Expertos.

Es importante mencionar que prácticamente en cualquier lenguaje pueden implementarse Sistemas Expertos; la diferencia radica en la facilidad que llegan a otorgar algunos de ellos, como por ejemplo PROLOG, C++, LISP, etc. Sin embargo, es necesario considerar, que mientras un lenguaje sea lo más especializado posible, se tendrán

mayores limitantes de caracter técnico para el desarrollo de sistemas.

Las herramientas son paquetes desarrollados en algún lenguaje de programación para desarrollar Sistemas Expertos. En estos, el usuario únicamente se encarga de especificar el dominio sobre el que va a operar el sistema.

Estos paquetes permiten la creación de Sistemas Expertos un tanto rígidos, pero que pueden ser implementados por personal sin grandes conocimientos de informática.

IV.2 LENGUAJES PARA EL DESARROLLO DE SISTEMAS EXPERTOS

IV.2.1 LENGUAJES IMPERATIVOS

Los lenguajes imperativos son aquellos que únicamente permiten una secuencia lineal en el programa, ya que no incorporan las facilidades que proporcionan el manejo de funciones o subrutinas.

Es preciso mencionar, que de acuerdo a lo anterior, estos

lenguajes prácticamente han desaparecido del ambiente informático actual ya que, entre otras cosas, no permiten aplicar con propiedad los preceptos de la programación estructurada. Incluso, lenguajes que anteriormente eran considerados como puramente imperativos, como el ensamblador, han implementado ya el uso de subrutinas en sus versiones más recientes.

Como es lógico suponer, este tipo de lenguajes no proporcionan facilidades para el desarrollo de Sistemas Expertos, ya que la construcción de éstos, tiene como consecuencia un mantenimiento poco eficaz, debido a la complejidad del programa.

IV.2.2 LENGUAJES FUNCIONALES

Los lenguajes funcionales son los más comunes e incorporan el uso de funciones que permiten un mejor control en la secuencia del programa. Ejemplos de ello son Pascal, C, LISP, etc.

Es importante mencionar, que LISP es el lenguaje funcional por excelencia, debido a que prácticamente, cada instrucción es una descripción de una función. Sin embargo, actualmente es muy utilizado el lenguaje C, debido a lo siguiente:

- Existen compiladores de C, para casi cualquier sistema

operativo.

- El código, una vez compilado es tan rápido como el ensamblador, a diferencia de los demás lenguajes.

- Permite implementar bases de conocimiento, de datos, motores de inferencia, reconocedores y otras rutinas, de acuerdo a la medida de la aplicación deseada.

La construcción de un Sistema Experto en lenguajes funcionales es muy empleada, debido a la capacidad de contar con estructuras de datos avanzadas, permitiendo la facilidad de otorgar un tratamiento simbólico a la información, lo que constituye una gran ventaja, para considerar su uso.

Otra razón para utilizarlos es la factibilidad de emplear algoritmos complejos como el que puede presentar la construcción de un motor de inferencia, así como las facilidades para el diseño de interfases de comunicación, sobre todo, las versiones más recientes.

Las desventaja que presenta la construcción de Sistemas Expertos en lenguajes funcionales, es la necesidad de instrumentar a nivel programación, todos los componentes que conforman a dicho sistema. Sin embargo, esta desventaja puede considerarse relativa, si tenemos en cuenta que nuestro sistema utilizará rutinas más precisas y transparentes, lo que no ocurre en un lenguaje como PROLOG, en donde la elección de soluciones es poco clara.

LISP

LISP (LIst Processing) es el lenguaje funcional más utilizado en el diseño de Sistemas Expertos, y de hecho uno de los primeros en el mundo de la Inteligencia Artificial. Creado por John McCarthy - uno de los pioneros de la Inteligencia Artificial- en 1960, este lenguaje está orientado básicamente, al procesamiento de listas.

La estructura de datos esencial en LISP, es una secuencia de elementos, denominada lista, formada de entidades indivisibles (átomos), como funciones, caracteres o números, o bien, otras listas. Las listas son esenciales en el diseño de Sistemas Expertos debido a su flexibilidad y en este lenguaje, el programador no necesita especificar por anticipado el número o tipo de elementos en una lista.

Las listas pueden ser utilizadas para representar un arreglo ilimitado de objetos y de reglas que requieran de ser procesados en nuestro sistema.

LISP permite el diseño de motores de inferencia poderosos, de una manera sencilla, aunque, su principal problema radique en la dificultad de implementar bases de conocimiento lo suficientemente flexibles. Sin embargo, las nuevas versiones del lenguaje, proporcionan herramientas que permiten la implementación de este módulo de una manera sencilla.

Originalmente, las primeras versiones de LISP fueron desarrolladas con un pequeño conjunto de funciones para la manipulación de listas. Estas funciones fueron realizadas de tal forma que permitieran la creación de nuevas funciones a partir de las ya existentes.

Actualmente, LISP tiene muchas funciones y características que facilitan el desarrollo de aplicaciones. Entre todas las versiones, sobresale COMMON LISP, que se ha ganado la aceptación de un estándar.

En el ámbito de las computadoras personales, la primera implementación para PC fue Golden Common LISP, de Golden Hill que apareció en 1984, y que todavía es ampliamente utilizado. La primera implementación en Macintosh, ExperLisp, apareció en el mismo lapso de tiempo. Otras implementaciones muy populares de LISP son: muLISP 90 de Soft Warehouse, ManoLISP de Microcomputer Systems y XLISP, que corren en una gran variedad de computadoras.

LOGO

Desarrollado en los años 60's por Papert en el MIT, como un lenguaje auxiliar en la educación para niños, LOGO en realidad puede considerarse como un subconjunto de LISP, y su utilización como un lenguaje de programación serio para el diseño de Sistemas Expertos, es muy factible.

LOGO es bien conocido por su capacidad de graficar: el usuario programa el movimiento del cursor (comúnmente llamado "tortuga"), para dibujar sobre la pantalla. Aparte de las gráficas, LOGO tiene un número de funciones para manipulación de listas que utilizan una sintaxis mucho más amigable que LISP; además cuenta con un generador de números aleatorios, que permite la simulación de procesos, característica importante de las técnicas de Inteligencia Artificial.

LOGO es un lenguaje compacto, amigable y sencillo de usar, y constituye un excelente vehículo para el estudio de técnicas de Inteligencia Artificial en PC. Se tienen muchas implementaciones de este lenguaje, en el que sobresalen ExperLOGO Plus de ExperTelligence y PC LOGO de Harvard.

IV.2.3 LENGUAJES ORIENTADOS AL OBJETO.

Los lenguajes orientados al objeto son lenguajes muy poderosos para el desarrollo de Sistemas Expertos. Su uso se hace cada vez más frecuente debido a la facilidad de resolver problemas complejos empleando un código fuente más compacto y más fácilmente manejable. La filosofía de programación empleada es diferente a los demás lenguajes, y básicamente se basan en la estructuración como una unidad conjunta de los datos y las funciones que las manipulan.

Los lenguajes orientados al objeto brindan la capacidad de implementar de manera directa la representación en base a objetos, ya que incorporan instrucciones propias para su creación y manipulación. La mayoría de estos lenguajes son capaces de reconocer las tres características fundamentales con que debe contar un sistema de representación en base a objetos:

1) La herencia, es una característica que permite la creación de objetos a partir de objetos previamente definidos, evitando la codificación de programa fuente innecesario. La herencia es útil en la creación de objetos específicos que toman las características generales de sus antecesores y además añaden algunos rasgos propios del objeto.

Ejemplo.

Si tenemos la clase AVION definida con los siguientes atributos y métodos.

```
Clase: Avión
Atributos: Alas, velocidad, asientos
Métodos: Asigna_valores, despegue, aterrizaje, vuelo
```

Es posible definir a la clase HELICOPTERO para que herede valores de AVION:

```
Clase  Helicóptero :hereda atributos de Avión:
Atributos propios  tipo_de_hélice, control_T
Método propio  asigna_nvo_valor
```

Si obj_helicoptero es un objeto de la clase helicóptero,la siguiente sentencia es válida:

```
obj_helicoptero velocidad = 500
```

2) Encapsulamiento de datos, se refiere a la manipulación de del código de datos de los objetos exclusivamente en el ámbito de las funciones que la conforman.

Ejemplo: En el ejemplo anterior, es posible encapsular las variables para evitar que sean accesadas en otros módulos ajenos a los de la clase avión, de la siguiente manera:

Clase: Avión
Atributos privados: llantas, velocidad, asientos
Métodos: Asigna_valores, despegue, aterrizaje, vuelo

Esto evita que los atributos de la clase avión sean accesados en secciones ajenas a las de los métodos de la clase

3) Polimorfismo, se refiere principalmente al empleo de funciones sin necesidad de especificar el tipo de datos al que pertenecen los argumentos Esto se logra con la implementación de diversas funciones que prácticamente realizan lo mismo aunque con diferentes tipos de datos como entrada.

Ejemplo: En un lenguaje que se precie de ser orientado al objeto es posible implementar más de una función con el mismo nombre. Supongamos una función asigna que pueda recibir como parámetros tipos de datos string (cadena), integer (entero) y real, para asignárle el resultado a uno de sus atributos.

Clase: Prueba_polimorfismo
Atributos: x,y
Métodos: asigna (char *a, char *b)
asigna (int a, int b)
asigna (float a, float b

Cada uno de los métodos implementados deben realizar la conversión del tipo de datos para poder procesarlo. El

compilador en cuestión (C++, Turbo Pascal, etc) determina en tiempo de ejecución el método que ha de ser ejecutado validándose para ello en el tipo de dato pasado como parámetro.

En la actualidad, existen numerosos lenguajes de programación orientados al objeto, entre los que destacan Smalltalk, C++, Turbo Pascal, Actor, etc.

SMALLTALK

En los inicios de los años setentas, investigadores del Centro de Investigación de Palo Alto de Xerox, construyeron un medio ambiente para que personas sin muchos conocimientos de informática pudieran construir aplicaciones complejas. Su diseño se fundamentó en los estudios realizados sobre el proceso de pensamiento, y así desarrollaron Smalltalk.

Smalltalk combina la programación orientada a objetos con una interfase gráfica de usuario y un estilo de programación interactivo. El medio ambiente de Smalltalk permite al usuario crear clases y objetos fácil y rápidamente.

La compañía Digitalk ha desarrollado Smalltalk/V para computadoras con DOS y Macintosh. Además, versiones para una amplia variedad de máquinas son producidas por el Departamento de Ciencia Informática de la Universidad del Estado de Oregón, en Estados Unidos.

C++

C++ fue creado en 1983 en los laboratorios Bell de la AT&T, y en la actualidad constituye uno de los lenguajes de programación orientados al objeto más poderosos. Es un lenguaje sucesor de C, e incorpora facilidades para manejo de objetos directamente desde su declaración.

Actualmente, las nuevas versiones tienen la capacidad de crear herencias múltiples, así como de usar punteros en los miembros de una clase.

Su aplicación en la programación de Sistemas Expertos es más factible, con relación a otros lenguajes, debido a un mayor número de programadores en C, que de LISP, PROLOG o Smalltalk, por ejemplo.

Una vez que se han definido los objetos en C++, éstos no interactúan con el resto del programa directamente, y no hay manera de adaptarlos a nuevos usos excepto modificando su definición.

Cada uno de los objetos en C++, independientemente de la clase donde sean utilizados, deberán ser inicializados por un método público definido en la clase a la que pertenecen. Además, es requisito de una buena programación, modificar los objetos

únicamente en el ámbito de los métodos que la conforman.

En general, las ventajas que C++ brinda en la realización de Sistemas Expertos son:

- Es ejecutable en una gran mayoría de máquinas tradicionales.
- Coexiste con sistemas operativos tradicionales.
- Tiene una gran velocidad de ejecución.

Las versiones más utilizadas de C++ son C++ Language System Release 2.0 y Turbo C++ de Borland Internacional Inc., que puede ser ejecutada en una amplia gama de computadoras personales.

IV.2.4 LENGUAJES DECLARATIVOS.

Los lenguajes declarativos, también llamados lenguajes de tipo lógico, [1] "Son aquellos en los que solamente hay que indicarle al programa, el objetivo que queremos demostrar, especificando en el programa el universo sobre el que debe demostrarlo y las reglas que puede utilizar".

Los lenguajes declarativos incorporan dentro de su estructura un mecanismo de inferencia con características perfectamente

[1] JUAN PABLO SANCHEZ Y BELTRAN.

establecidas y no variables; esto permite que no exista la necesidad de diseñarlo, pero a la vez representa una desventaja, ya que la elección de soluciones, no es muy transparente al programador, si éste no tiene un conocimiento profundo del funcionamiento del mecanismo implementado en el lenguaje.

Dos de los lenguajes declarativos más utilizados en el diseño de Sistemas Expertos son PROLOG y OPS5.

PROLOG

PROLOG (PROgramación LOGica), es uno de los lenguajes más populares en el diseño de Sistemas Expertos y fue creado por Colmerauer entre los años 60 y 70. Clocksin y Mellish en la Universidad de Edinburgh, perfeccionaron su sintaxis y hasta el momento, su versión, denominada C&M Sintax ó Edinburgh Sintax, es aceptada como un estándar.

Entender su funcionamiento puede ser complicado, especialmente si siempre se ha trabajado con lenguajes tradicionales, pero una vez que se le domina, es puro y amigable.

PROLOG cuenta con un mecanismo de encadenamiento hacia atrás, denominado "backtracking" o vuelta atrás, y de una búsqueda en profundidad para la resolución de problemas.

ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

La representación de conocimiento empleada es en base a reglas de producción.

Las nuevas versiones del lenguaje, como PDC PROLOG de PROLOG Development Center, pueden ser utilizados incluso, como lenguajes de propósito general debido a la gran variedad de instrucciones que incorpora. En PDC PROLOG, es posible la utilización de estructuras de datos dinámicas y del manejo de apuntadores, además del chequeo de tipos. El proceso de unificación es desarrollado de una manera sencilla. Además, incorporan facilidades en el manejo de listas, gráficas y reglas que pueden ser almacenados en una base de datos multiusuario, incluso. Cuenta con herramientas para ser utilizado en redes, asi como para interactuar con bases de datos desarrollados en muchos paquetes comerciales.

Además del PDC PROLOG (o Turbo PROLOG), existen otras implementaciones para DOS: Arity PROLOG de la Arity Corporation, Quintus PROLOG de Quintus Computer Systems. Para Macintosh, Advanced Artificial Inteligence Systems produce AAIS PROLOG, y ExperTelligence, ExperProlog II, que es una versión de un grupo de Colmerauer.

OPS5

En los años 70's, Charles Forgy desarrolló OPS5, un lenguaje de tipo declarativo que utiliza una representación de conocimiento en

base a reglas de producción. Su trabajo se basó en estudios realizados en la Universidad Carnegie Mellon sobre el proceso de pensamiento, realizado por reconocidos pioneros de la Inteligencia Artificial: Herbert Simon y Allen Newell.

En OPS5 se han realizado diversos Sistemas Expertos comerciales, entre ellos XCON, usado por la Digital Equipment Corporation para configuración de sus computadoras, y otros para el estudio del espacio.

OPS5 almacena los datos en la memoria de trabajo, y las reglas en la memoria de producción. Estas reglas son almacenadas de manera independiente y en cualquier orden. Si los datos en la memoria de trabajo cumplen con el conjunto de condiciones de las reglas en cuestión, las acciones de éstas son realizadas.

Algunas versiones de OPS5 tienen la capacidad de crear nuevas reglas, permitiendo el desarrollo de sistemas que aprenden, constituyendo un factor importantísimo para considerar su utilización.

El mecanismo de inferencia implementado en OPS5 es el de encadenamiento hacia adelante (forward chaining).

Originalmente diseñado para mainframes, OPS5 está ya disponible en computadoras personales: J Soft and Dynamic Master Systems

desarrolla lenguajes que funcionan sobre la plataforma de DOS, y ExperTelligence, produce versiones para Macintosh.

IV.2.5 UNION DE LENGUAJES

La fase de programación de un Sistema Experto, puede desarrollarse en varios lenguajes, para aprovechar las mejores características que brindan cada uno de ellos. Algunas de las más importantes compañías de software han puesto especial énfasis en el desarrollo de utilerías que posibiliten la interacción de distintos lenguajes. Esto se conoce como unión de lenguajes.

La unión puede ser factible de diversas formas, las mas comunes son:

- Mediante la unión de módulos previamente compilados, es decir, de archivos OBJ. Estos archivos pueden ser creados por una gran mayoría de compiladores. La unión ocurre en el proceso de generación del archivo ejecutable (linking), y para ello pueden utilizarse algunos de los "linkers" más conocidos (RTLink, TurboLink, PLink86, etc).

- Mediante la utilización de lenguajes híbridos, que toman las mejores características de algunos lenguajes para unirlos en uno

solo. Como ejemplos, están el Lislog, que es una unión de Lisp y Prolog, ó modula-2, que es una unión de Pascal y Prolog.

- Mediante la extensión del lenguaje, es decir, incorporando funciones, que quizá, no entran en el contexto del mismo, pero que son de gran ayuda en la programación. Un ejemplo es el uso de funciones matemáticas en Prolog y VP Expert.

IV.3 HERRAMIENTAS PARA SISTEMAS EXPERTOS

IV.3.1 ENTORNOS DE DESARROLLO

Los entornos de desarrollo son un conjunto de utilerías que hacen posible la creación de Sistemas Expertos con pequeños conocimientos sobre el tema, a diferencia de los sistemas vacios. Básicamente vienen implementados con:

- Un motor de inferencia de cualquier tipo. Inclusive algunos muy poderosos, vienen integrados con varios.

- Una formato especifico de representación del conocimiento.

- Un editor, para que se proporcione la base de conocimiento.

- Utilerías para la búsqueda de redundancias, inconsistencias y

contradicciones en la base de conocimiento.

- Utilerías para dar al usuario final explicaciones y justificaciones a las soluciones encontradas por el motor de inferencia.

- Utilerías para diseño de interfases gráficas de usuario.

Aunque este tipo de herramientas no han sido desarrolladas, en general, a partir de un Sistema Experto, su uso se centra sobre dominios limitados de aplicación.

Algunas de las herramientas comerciales más conocidas son VP Expert, Arity Expert System, Expert OPS5, Nexpert, etc.

VP EXPERT

VP Expert es un entorno de desarrollo diseñado por Paperback Software Internacional, que cuenta con un mecanismo de encadenamiento hacia atrás, permitiendo la creación de bases de conocimiento con una representación en base a reglas de producción, a partir incluso, de bases de datos, hojas de cálculo o archivos tipo texto.

Una aplicación en VP Expert cuenta con 3 secciones principales:

- La sección de acciones (ACTIONS), se ejecuta al correr el programa. En ésta se especifican, los objetivos que tiene que cumplir nuestro sistema.

- La sección de reglas (RULES), en la que se definen el conjunto de reglas a emplear.

- La sección de variables, en donde se definen las variables, cuyos valores serán preguntados al usuario, especificando el rango permitido, si así se requiere.

VP Expert además, cuenta con un mecanismo para información de justificaciones, sobre las soluciones a las que se han llegado.

IV.3.2. SISTEMAS VACIOS

Son conocidos también como "Shells", y son sistemas a los que únicamente se les da el conocimiento en un formato específico para desarrollar Sistemas Expertos. Su utilización puede ser realizada por personas sin conocimientos en Sistemas Expertos, debido a la utilización de interfases gráficas excelentes y claramente legibles.

En realidad, es difícil hacer una distinción entre entornos de desarrollo y sistemas vacios, comúnmente la distinción se realiza

en base a la dificultad para manejarlo: si puede realizarlo una persona sin conocimientos en Sistemas Expertos, es un sistema vacío, de lo contrario, es un entorno de desarrollo. Los sistemas vacios, son además más caros.

Entre los 'shells' mas populares tenemos a Guru de Micro Data Base System, Icarus de Icarus Corporation, Mentor de Mentor Corporation, etc.

GURU

Desarrollado por Micro Data Base System, Guru es uno de los sistemas vacios más populares actualmente. Es una herramienta muy poderosa para crear Sistemas Expertos de una manera sencilla y amigable.

Utiliza un mecanismo de encadenamiento hacia atrás, así como un sistema de representación de conocimiento en base a reglas.

Guru es considerado un sistema vacío debido a la utilización de una interfase bastante accesible y amigable, permitiendo su utilización a personas sin conocimientos en Sistemas Expertos.

Guru cuenta con lo siguiente:

- Un editor para accesar las reglas, sus justificaciones, sus factores de certidumbre, sus prioridades, así como comentarios.

- Un editor para accesar las variables a utilizar, indicando el nombre del identificador, su tipo y su rango de valores.

- Un sistema de justificaciones.

- Un sistema detector de inconsistencias y errores.

- Un sistema para desarrollar interfases de comunicación de una manera sencilla.

Por todas estas características, Guru es considerado uno de las mejores herramientas para desarrollar Sistemas Expertos.

CAPITULO V

ORDENADORES EN EL DESARROLLO DE LOS SISTEMAS EXPERTOS

V.1 INTRODUCCION

Los Sistemas Expertos pueden ser desarrollados en una gran variedad de máquinas, sin embargo, se han desarrollado computadoras dedicadas exclusivamente a su aplicación. Las ventajas son muchas, pero entre todas ellas sobresale la velocidad con que se ejecutan.

Entre las principales clasificaciones de computadoras especializadas en el desarrollo de Sistemas Expertos destaca la siguiente:

- Miniordenadores
- Estaciones de Trabajo
- Máquinas Simbolicas
- Ordenadores Paralelos

V.2 MINIORDENADORES

Las minicomputadoras están basadas en procesadores de 16 y 32 bits, y generalmente no han sido diseñadas para procesos simbólicos -requeridos en Sistemas Expertos-, sino más bien, numéricos.

El desarrollo de Sistemas Expertos en minicomputadoras es ampliamente requerida, debido a la gran difusión que tienen, y a la existencia -cada vez mayor- de una extensa variedad de lenguajes y herramientas en el mercado. Sin embargo, por las características de la mayoría de ellos, las aplicaciones están muy limitadas debido a la lentitud que origina el manejo de una tecnología no apta para el tratamiento de información requerida por los sistemas desarrollados en base a mecanismos de Inteligencia Artificial, como pueden ser procesadores, memorias, etc.

Entre las principales minicomputadoras utilizadas en el desarrollo de Sistemas Expertos tenemos a las computadoras personales, basadas en microprocesadores 80386 y 80486 de Intel,

así como aquellas que cuentan con el 68000 y 68020 de Motorola, como las Apple y Commodore.

V.3 ESTACIONES DE TRABAJO

Las estaciones de trabajo (Work Station), son computadoras basadas en procesadores de 32 bits y diseñadas para aplicaciones específicas en algunas áreas: Inteligencia Artificial, CAD CAM, Hipermedia, modelaje de sólidos, animación por computadora, etc.

Están basadas en potentes microprocesadores basados en la tecnología RISC.

La capacidad de memoria que tienen es considerablemente grande, ya que como mínimo poseen 4 MB de RAM, y un almacenamiento secundario prácticamente ilimitado.

Las estaciones de trabajo en su mayoría, son mucho más veloces que los miniordenadores, debido al uso de procesadores más potentes y sobre todo, de una configuración más acorde a la utilización para la que fueron diseñadas. Algunas estaciones poseen coprocesadores con manejo simbólico de la información (Personal Symbolic Computer de Amaia, por ejemplo), que agilizan aún más la velocidad de ejecución de los sistemas. En algunos casos proporcionan una

velocidad similar al de las máquinas simbólicas, con la ventaja de un precio mucho menor.

La mayor desventaja de las estaciones de trabajo son la incompatibilidad entre marcas distintas propiciado por la carencia de un estándar a seguir.

Entre las principales estaciones de trabajo se encuentran: PC MX-2 de IF Lavarde, AI VAX Station II de Digital Equipment, la 4404 de Tektronix, HP 9000 Series 300 y 800 de Hewlett Packard, Explorer LX de Texas Instruments, RT 6160 de IBM, etc.

V.4 MAQUINAS SIMBOLICAS

Las máquinas simbólicas son computadoras diseñadas especialmente para el tratamiento de símbolos. Están equipados con microprocesadores simbólicos, que tienen la característica de poseer uno o varios compiladores integrados. Comúnmente se les denomina máquinas LISP, por ser el primer lenguaje implementado en la mayoría de los equipos.

En el proceso de compilación de sistemas en una máquina simbólica, se genera un microcódigo, directamente ejecutable por el microprocesador, lo que permite una gran velocidad en la

aplicación.

Algunos autores efectúan una clasificación de este tipo de máquinas en relación al número y funcionamento interno de los microprocesadores:

- **Máquinas tipo M**, donde únicamente existe un microprocesador con soporte para uno o más lenguajes. Es la configuración más popular, basada en la arquitectura secuencial propuesta por Von Newman en 1945.

- **Máquinas tipo S**, donde existen varios microprocesadores con un conjunto de instrucciones específicas cada uno de ellos. Estas máquinas poseen una arquitectura paralela con un enfoque en "pipeline".

- **Máquinas tipo P**, donde existen varios microprocesadores de igual manufactura y que realizan cada uno de ellos la misma acción. Estas máquinas poseen una arquitectura paralela con un enfoque concurrente.

La mayoría de las máquinas simbólicas tienen microprocesadores de 32 bits de palabra; memoria RAM de 4 hasta 128 MB y almacenamiento secundario hasta de 3.5 GB.

Las primeras máquinas simbólicas fueron construidas en los años

70 en los laboratorios del Instituto Tecnológico de Massachussets. Actualmente entre las mas populares se encuentran:

- **TI Explorer**, con microprocesador de 32 bits de palabra y soporte para Zetalisp, Common Lisp, Interlisp y Prolog.

- **Symbolics 3600**, con microprocesador de 36 bits de palabra y soporte para Zetalisp, Common Lisp, Interlisp y Prolog.

- **LMI Lambda**, con microprocesador de 40 bits y soporte para Zetalisp, Common Lisp, Interlisp y Prolog. Esta computadora es muy utilizada debido a que soportan la plataforma UNIX, permitiendo el desarrollo de sistemas abiertos.

- **Xerox 1100**, con microprocesador de 16 bits y soporte para Common Lisp, Interlisp, Smalltalk y Prolog.

- **PSI (Personal Sequential Inference)**, diseñada como objetivo inicial en el Proyecto de Quinta Generación de Computadoras. Tiene soporte para Prolog.

La principal ventaja de las máquinas simbólicas para desarrollo de Sistemas Expertos es la velocidad con que se ejecutan, así como la factibilidad de implementar aplicaciones sofisticadas debido a la existencia de hardware y software específico. Sin embargo los elementos principales para su poca difusión es el costo elevado que

poseen y la incompatibilidad que existe entre ellas por la utilización de una gran diversidad de microprocesadores.

V.5 ORDENADORES PARALELOS

Los ordenadores paralelos son aquellos que utilizan una arquitectura paralela en el tratamiento de la información en base a la utilización de varios microprocesadores.

Su utilización es muy frecuente a pequeña escala, en los miniordenadores y estaciones de trabajo, con la implantación de coprocesadores, y a regular escala en las máquinas simbólicas tipo S y P.

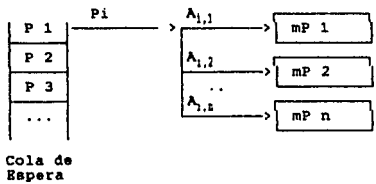
Existen en la actualidad muchos proyectos orientados al diseño de estas máquinas, entre los que destaca el proyecto de Quinta Generación de Computadoras.

V.5.1 PARALELISMO

El paralelismo es una técnica empleada para aprovechar al máximo el uso de los microprocesadores disponibles. Básicamente puede

ocurrir de 2 formas:

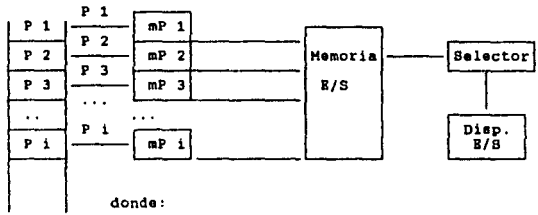
a) Pipeline, donde únicamente un proceso es tomado de la cola de espera para su ejecución. Los microprocesadores ejecutan la parte de acción que les corresponde. Un diagrama es el siguiente:



donde:

Pi,	Proceso i
Ai,x	Acción x de proceso i
mP k	Microprocesador k
n	Número de microprocesadores

b) Concurrencia, donde son tomados simultáneamente de la cola de espera tantos procesos como microprocesadores existan, para su ejecución. Un diagrama es el siguiente:



donde:

i es el número de microprocesadores
Pj es el proceso j
mP j es el microprocesador j

La memoria es utilizada para evitar colisiones en la información direccionada a los dispositivos de E/S. De esta manera es posible usar un selector de datos para la transmisión.

V.5.2 PROYECTO FGC

El proyecto de Quinta Generación de Computadoras (FGC, por sus siglas en inglés) anunciado por los japoneses en 1981 ha despertado un gran interés en los desarrolladores de Sistemas Expertos debido a la posible aparición de equipos que optimicen la ejecución de estos sistemas.

Básicamente, se planea la construcción de una máquina simbólica PROLOG con arquitectura paralela, además del diseño de una gran

variedad de soporte (comunicación, software, etc) que permitan una amplia gama de posibilidades en una sola computadora.

Como se mencionó anteriormente, se ha desarrollado como objetivo preliminar, la computadora PSI, una máquina simbólica PROLOG con arquitectura secuencial.

El proyecto se ha prolongado más tiempo de lo previsto, la comunidad informática sigue en espera de lo prometido, sin embargo, la liberación de estas máquinas, está cada vez más cercana.

V.6 PORTABILIDAD

La portabilidad de sistemas, es la capacidad de diseñar software sobre equipos que puedan ser manejados fácilmente debido a su contextura física.

La portabilidad cada vez es más importante debido a la necesidad de utilizar Sistemas Expertos en áreas donde difícilmente es factible la utilización de grandes equipos de cómputo.

Este es un punto a favor en el desarrollo de Sistemas Expertos en una gran mayoría de miniordenadores y estaciones de trabajo, debido a la gran portabilidad que poseen.

En la actualidad, existen muy pocos Sistemas Expertos portables, aunque se espera un auge considerable con el advenimiento de tecnología cada vez más compacta, basada en poderosos microprocesadores.

CAPITULO VI

CONSTRUCCION DE SISTEMAS EXPERTOS

Antes de analizar la metodología de construcción para desarrollar Sistemas Expertos, es necesario hacer énfasis en el desarrollo tradicional de software.

La metodología más comúnmente utilizada para desarrollo de software se basa fundamentalmente en 5 aspectos: análisis del problema, diseño del sistema, instrumentación, prueba y mantenimiento. Interaccionan como lo muestra el diagrama de la figura 6.1.

El principal problema de esta metodología consiste en la secuencia lineal de las etapas[1], que da como consecuencia un seguimiento inflexible en cada una de las secciones.

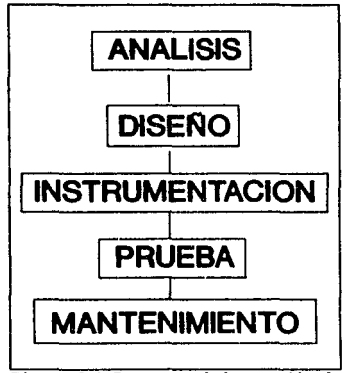


Figura 6.1 Fases en el desarrollo de sistemas tradicionales.

Lo anterior ocurre debido a la dificultad de entender completamente lo que se requiere en las etapas iniciales del sistema, siendo necesario en ocasiones redefinir nuevos objetivos. [2]"La metodología tradicional no contempla la aceptación de nuevos requerimientos en etapas avanzadas".

Lo anterior justifica la creación de una metodología que si permita posibles iteraciones en cualquier parte del desarrollo de sistemas, que faciliten la construcción de aplicaciones más eficientes.

---

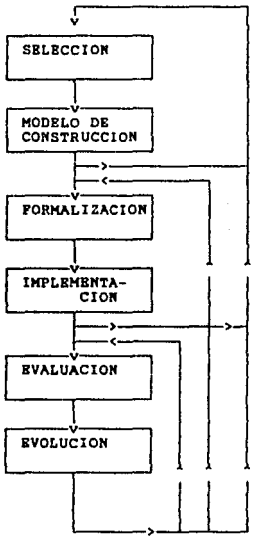
[1] Unicamente se pasa a una etapa posterior si la anterior se ha cumplido totalmente.

[2] RICHARD FAIRLEY

La metodología de construcción para Sistemas Expertos contempla estas mejoras, como se verá en la siguiente sección.

VI.1 METODOLOGIA DE LA CONSTRUCCION

La metodología que se ha desarrollado para facilitar la construcción de Sistemas Expertos consiste, básicamente en seis etapas: Selección del problema, elaboración del modelo de construcción, formalización, implementación, evaluación y evolución a largo plazo. Interactúan como lo muestra el siguiente diagrama:



Puede observarse que esta metodología si permite la revisión de una etapa determinada en cualquier fase de desarrollo, para contemplar nuevos objetivos, o para hacer cambios en los planteamientos iniciales.

VI.1.1 SELECCION DEL PROBLEMA

Consiste en la investigación del dominio de la aplicación, que permitirá realizar la selección y análisis de problemas que serán considerados en el desarrollo del sistema. Consiste de las siguientes etapas:

Investigación de la aplicación. Tiene como objetivo la familiarización preliminar del Ingeniero de Conocimiento con el dominio de la aplicación a tratar, y ello puede conseguirse con entrevistas a expertos y lectura de bibliografía adecuada principalmente.

Selección de problemas. El Ingeniero de conocimiento genera una lista de posibles problemas a considerar en el Sistema Experto. Esta selección se hace comúnmente utilizando varios criterios, como pueden ser la necesidad de aplicar conocimientos expertos en el problema, la viabilidad de contar con personal experto calificado en la rama en cuestión, la aceptación de que el sistema produzca respuestas probables en ocasiones y contar con un tiempo prudente.

Análisis de problemas. Una vez que se han generado los posibles problemas a considerar en la construcción del Sistema Experto, se realiza un análisis detallado de cada uno de ellos, para aceptarlos como candidatos finales.

Este análisis comprende la posible aplicabilidad del problema, la disposición de profesionales expertos, la delimitación del problema y el análisis de costo/beneficio.

La aplicabilidad del problema consiste en la verificación de la escasez y la demanda del conocimiento en el dominio donde se planea desarrollar la aplicación. Además se analiza que el problema realmente requiera de un razonamiento de tipo simbólico, y que la resolución de los problemas sea similar a la que se efectúa en forma real.

El análisis costo/beneficio realizado tiene como objetivo conocer la rentabilidad del Sistema Experto. El estudio se hace considerando factores tales como el salario del experto y del ingeniero de conocimiento, así como el costo del equipo de cómputo y el software a utilizar.

VI.1.2. MODELO DE CONSTRUCCION

Una vez realizada la selección de los problemas, se efectúa un pequeño prototipo del Sistema Experto a desarrollar. Esto tiene como objetivo, familiarizarnos más con el medio ambiente que rodea al sistema, así como tomar las primeras decisiones sobre el motor de inferencia, el tipo de representación de conocimiento y el

lenguaje o herramienta a utilizar. En esta fase se efectúan los siguientes pasos:

i) Adquisición de conocimiento inicial, en la que el ingeniero de conocimiento continúa haciendo investigaciones sobre el dominio de la aplicación, en forma cada vez más específica.

ii) Modelo de consulta, en donde se determina el formato en que se desarrollarán las consultas y los resultados.

iii) Selección del motor de inferencia, representación del conocimiento y el lenguaje o herramienta a utilizar, en donde el ingeniero de conocimiento selecciona las formas más apropiadas de los elementos que conforman al Sistema Experto de acuerdo a la aplicación, y a los recursos con que se cuente, como se describió en los capítulos anteriores.

iv) Implementación y prueba del prototipo, en donde el ingeniero de conocimiento desarrolla un pequeño sistema que cumpla con los requerimientos anteriores y que resuelva algunos ejemplos de problemas específicos. Se procede a efectuar pruebas hasta encontrarlo óptimo y continuar con las siguientes secciones, de lo contrario se hace una regresión a etapas anteriores para hacer correciones en donde se hayan detectado posibles errores.

VI.1.3. FORMALIZACION

Esta fase tiene como objetivo realizar una definición más detallada de los problemas, que permita efectuar una planeación precisa de las siguientes etapas. La formalización consiste de:

i) Definición detallada del problema. La realiza el ingeniero de conocimiento debido a que en esta fase tiene una idea más clara sobre el sistema, sus objetivos y restricciones.

ii) Diseño. En esta etapa, el ingeniero de conocimiento busca adecuar los análisis y planteamientos anteriores en representaciones que puedan ser implementados en un sistema computacional. En esta fase pueden utilizarse los conceptos del desarrollo tradicional, sin embargo es necesario afirmar que comúnmente se efectúan muchas iteraciones debido a que todavía no se tiene una idea muy clara de los niveles de detalle de los elementos del sistema. Deben tomarse en cuenta tres conceptos fundamentales en el diseño:

a) Abstracción, para especificar en forma cada vez más precisa las propiedades conceptuales con que deberá contar el motor de inferencia, la base de conocimiento y las interfases de comunicación (estructura de datos, tipo de rutinas requeridas, etc.).

b) Estructuración, [1]"para permitir que un sistema sea definido en términos de unidades más pequeñas y manejables con una clara definición de las relaciones entre las diferentes partes del sistema".

c) Modularidad, para que el sistema consista en unidades claramente definidas y manejables con las interfases claramente definidas entre los diversos módulos.

Una notación de diseño que recomendaría para bases de conocimiento es la de cartas de estructura modificada ya que no necesita de especificaciones de control, además que son directamente implementables. Un ejemplo es:

ENTRADA	SALIDA
Si Temperatura >40 y Peso < 40	Persona con cólera probable

Para el diseño del motor de inferencia y las interfases de comunicación son ampliamente utilizadas las técnicas de diseño de refinamiento por pasos de Nicolas Wirth, debido a su naturaleza de

[1] RICHARD FAIRLEY

contemplar la iteración de manera automática.

El refinamiento por pasos es una técnica cuyo objetivo es la descomposición de especificaciones de alto nivel en niveles cada vez más elementales. Es conocido también como "desarrollo a pasos de un programa". Un ejemplo es el siguiente:

Supóngase que se desea diseñar una función que busque en la base de conocimiento un objeto que cumpla con determinadas características:

Paso 1.

```
BUSCA()
   STRUCT atributo *q; /* estructura de atributos deseada */
  {
    define estructura de datos locales;

    hasta que se prueben todos los objetos, hacer:

      apunta=base[i].apu_atributos;
      Si las listas de atributos coinciden entonces
        printf("El objeto es %s",base[i].nombre;
        return;
      Fin_Si;

    Fin_hasta;
    printf ("Ningún objeto encontrado");
  }
```

Paso 2.

```
BUSCA()
  STRUCT atributo *q;
 {
   int i;
   struct atributo *p;

   for (i=0, i <= tamaño_base, i++)
     {
       p=base[i].apu_atributos;
```

```
        if coinciden (p,q)
          {
            printf("El objeto es %s ",base[i].nombre);
            return;
          }
      }
    printf("No hay objeto con esos atributos");
  }
```

Debemos enfatizar que un buen diseño hará posible que la etapa de implementación sea prácticamente directa.

iii) Planeación del proyecto, en donde se efectúa un presupuesto del sistema para conocer si es factible continuar con la construcción del Sistema Experto, así también se realiza la asignación de tareas específicas a cada uno de los elementos que conforman al grupo de desarrollo, en caso de sistemas muy grandes.

iv) Planeación de la prueba, en la que se realiza una lista de las pruebas que serán aplicadas al Sistema Experto para verificar su validez. Para pruebas óptimas es necesario que el ingeniero de conocimiento haya delimitado perfectamente el alcance del problema para efectuar examenes en los límites del dominio, que permitan inspeccionar el desenvolvimiento del sistema en los estados máximos y mínimos.

v) Planeación del producto, tiene como objetivo desarrollar la forma en que se hará la presentación inicial del sistema a personas allegadas, para posteriormente enviar a lugares estratégicos copias

beta[1] que permitan hacer modificaciones al sistema, mediante recomendaciones.

vi) Planeación del soporte, en esta fase el grupo de desarrollo estudia las formas en que se proporcionará el soporte a los usuarios, así como los encargados de llevarla a cabo.

vii) Planeación de la implementación, en donde se determina la forma en que se efectuarán los procesos en la etapa de implementación, como la adquisición de conocimientos por ejemplo.

El proceso de formalización hace que el ingeniero de conocimiento tome en cuenta todas las etapas en la construcción del Sistema Experto, para formar aplicaciones que satisfagan completamente al usuario.

VI.1.4. IMPLEMENTACION

En la fase de implementación se construye los componentes del Sistema Experto en base a todas las exposiciones anteriores. La implementación consta de lo siguiente:

[1] Copias del sistema susceptible de ser mejoradas antes de distribuirlas para uso general.

ii) Adecuación al modelo de construcción, en donde el motor de inferencia y la base de conocimiento del prototipo anterior son revisados para tratar de que cumplan con las especificaciones realizadas en la etapa de diseño. Así también, el ingeniero de conocimiento y el experto hacen una revisión a la base de conocimiento para que esta trate de contener elementos de conocimiento de bajo nivel[5] que posibiliten la flexibilidad del sistema. En esta etapa el ingeniero de conocimiento realiza la partición de la base de conocimiento, en donde ésta se modula permitiendo que las interrelaciones entre las diferentes estructuras de conocimiento sean lo más flexible (cohesión) y evitando al máximo el numero de conexiones entre ellas (acoplamiento).

iii) Implementación del motor de inferencia, la base de conocimiento y las interfases de comunicación. En esta fase son instrumentados a nivel programación los componentes que conforman al Sistema Experto utilizando el lenguaje o herramienta seleccionada. Es necesario hacer mención que estos componentes pueden ser implementados en lenguajes diferentes, siempre y cuando estos soporten las interfases necesarias para su posterior integración.

iii) Revisión de la base de conocimiento final, se realiza con el fin de evitar las redundancias e inconsistencias que afecten a la

[5] Son aquellos conocimientos de tipo esenciales y que son la base para construir conocimientos más complejos o de alto nivel.

base de datos. Comúnmente se efectúa ejecutando el plan de pruebas realizado en la fase de formalización.

iv) Desarrollo de software auxiliar, para posibilitar la utilización de bases de datos, hojas de cálculo u otros implementos en el Sistema Experto, que permitan una mayor potencia y un ahorro considerable de tiempo (en captura de datos, por ejemplo). El software es desarrollado siguiendo las premisas de construcción de los sistemas tradicionales.

v) Integración y verificación interna, donde los componentes del Sistema Experto son integrados para formar una unidad conjunta. Así también el ingeniero de conocimiento en coordinación con el experto verifican su funcionamiento correcto para que la etapa de implementación finalize.

VI.1.5. EVALUACION

En la evaluación con respecto a la funcionalidad de un Sistema Experto, comúnmente se efectúa un test de Turing modificado: "Si el sistema es capaz de dar una serie de respuestas aproximadamente como las que proporcionaría un experto humano, entonces se trata de un verdadero Sistema Experto".

El test de Turing modificado es llevado a cabo de diversas formas, sin embargo la más utilizada es en base a encuestas a diversos expertos.

VI.1.6. EVOLUCION

La fase de evolución es una fase que por lo general nunca termina, ya que mediante ella es posible adecuar el Sistema Experto hacia una funcionalidad cada vez mejor dentro de un medio ambiente variable.

Entre sus objetivos está la adición de nuevos conocimientos que permitan bases de datos cada vez más completas, así como la posible inclusión de nuevos dominios.

Otra cuestión muy ligada a la funcionalidad del Sistema Experto es la aparición de nuevas versiones del lenguaje o lenguajes en el que fueron implementados los componentes del Sistema Experto. Las nuevas versiones por lo general permiten la optimización de nuestro programa y un ahorro de código considerable.

Una implementación llevada a cabo en forma correcta, permitirá que la etapa de evolución sea casi inexistente.

La metodología de construcción para sistemas expertos es muy

flexible y quizá en un futuro no muy lejano sea la base para el desarrollo de sistemas con arquitectura tradicional.

CAPITULO VII

DESARROLLO DE HERRAMIENTAS PARA SISTEMAS EXPERTOS

En el presente capítulo explicaré las principales herramientas que proporcionan tres de los lenguajes de programación más ampliamente utilizados para el desarrollo de Sistemas Expertos en computadoras personales: Pascal, Prolog y C++. Las versiones que detallaremos son las que proporciona la compañía Borland debido a que son las más comunes en el mercado, aunque sin embargo no debemos dejar de mencionar la existencia de otras versiones de estos lenguajes: C++ Language System, Arity Prolog, ExperProlog, etc.

TURBO PASCAL

Las nuevas versiones de Turbo Pascal incluyen dentro de sus rutinas, un conjunto de características que facilitan la utilización de la programación orientada a objetos. Explicaremos algunos de ellos.

La definición de una clase se efectúa en la sección de tipos de Pascal (TYPE), de una forma muy similar a como se definen los registros (RECORDS):

```
TYPE cola = OBJECT
              numeros : array [1..100] of integer;
              principio, fin : integer;
              procedure iniciar;
              procedure anadir ( elemento : integer);
              function quitar : integer;
            END;
```

Observe como la palabra reservada OBJECT en realidad no define un objeto sino una clase. Los objetos de la clase creada son declarados en la sección de variables (VAR) :

```
VAR obj_cola : cola;
```

Hasta aquí se ha creado un objeto, sin embargo todavía es necesario la implementación de los métodos que conforman a la clase. Esto se realiza en la sección de declaración de procedimientos y funciones:

```
PROCEDURE cola.iniciar;
  BEGIN
     principio:=0;
     fin:=0;
  END;
```

Los métodos pueden ser accesados en cualquier sección del programa fuente mediante el operador punto. El siguiente código es perfectamente válido:

```
BEGIN  { ..... PRINCIPAL ..... }
 ...
 obj_cola.iniciar;
 obj_cola.anadir (4);
 n := obj_cola.quitar;
 ...
END
```

Analizaremos brevemente la manera en que implementa Turbo Pascal tres de las características más importantes de un lenguaje orientado al objeto: El encapsulamiento de datos, la herencia y el polimorfismo.

1) Encapsulamiento de datos. Es una característica necesaria en todo lenguaje orientado al objeto, y Turbo Pascal en su versión 5.5 no la implementa en forma correcta como la realizan otros lenguajes (C++, por ejemplo).

Turbo Pascal realiza un encapsulamiento de tipo público a los elementos de un objeto (código de datos y métodos). Esto significa que el código de datos puede ser manipulado no únicamente en el ámbito de los métodos que lo conforman sino también en cualquier

sección del programa fuente, constituyendo un manejo ineficiente del concepto de encapsulamiento. Considerando el programa principal del ejemplo anterior, la sentencia siguiente es perfectamente válida:

```
BEGIN { PRINCIPAL }
 ........
    writeln ('posicion final: ',obj_cola.fin)
 ........
END.
```

Las versiones posteriores de Turbo Pascal a la examinada (5.5) deben considerar el empleo de un mecanismo que permita un manejo más eficiente del concepto de encapsulamiento, sobre todo en el código de datos, aunque sería bienvenido la posibilidad de encapsular en forma privada a algunos métodos que no tendrían razón de utilizarse en secciones ajenas a los métodos públicos de la misma clase a la que pertenecen.

2) Herencia. Turbo Pascal lleva a cabo la herencia de una manera muy sencilla. Consideremos la siguiente definición de la clase **vehiculos**:

```
TYPE vehiculos = OBJECT
                   ruedas : integer;
                   peso   : real;
                   function muestra_ruedas : integer;
                   function muestra_peso : integer;
                   procedure asigna (r: integer; p:real);
                 END;
```

La creación de una clase denominada **vehículos de carga** requiere

de las mismas características que definen a la clase vehiculos y además algunas que sean propias de ellos. Es necesario entonces que vehiculos de carga herede características de vehiculos. Este proceso también se realiza en la sección de tipos:

```
vehic_carga = OBJECT ( vehiculos )
                tipo_carga : string [80];
                procedure muestra_carga;
                procedure asigna_carga (c :string [80]);
              END;
```

Observe como el uso de los paréntesis indica al compilador que la clase vehic_carga adquirirá características de la clase vehiculos. Un objeto de la clase heredada se crea de igual forma:

```
VAR objeto2 : vehic_carga;
```

La forma en como puede accesar este objeto a los atributos que le fueron heredados, es también mediante el operador punto:

```
objeto2.ruedas := 4;
```

La implementación que hace Turbo Pascal para efectuar esta característica es elemental, pues todavía no es posible definir clases derivadas que hereden características de múltiples clases (herencia múltiple).

3) Polimorfismo. Consiste básicamente en que [1]"un nombre se puede

[1] HERBERT SCHILDT

utilizar para especificar una clase genérica de acciones". Se utiliza principalmente para desarrollar diversas rutinas que generalmente realizan lo mismo, excepto que con diferentes tipos de datos. Así se logra que tengamos la impresión de que una sola función puede realizar el mismo proceso sin importar el tipo de sus argumentos, aunque en realidad se ha implementado una función para cada caso.

La implementación que Turbo Pascal hace del polimorfismo es en base a métodos virtuales[1]. Supongamos que mostrar va a ser un método virtual. El método virtual será reconocido únicamente en el ámbito de la clase donde fue creado, por consecuencia las clases derivadas implementarán su propia versión del método virtual:

```
TYPE vehiculos = OBJECT
            ruedas: integer; peso: real;
            procedure mostrar ( r: integer, p:real); virtual;
     END;

    vehic_carga = OBJECT (vehiculos)
                  tipo_carga : string [80];
                  procedure mostrar (tc :string [80]); virtual;
                END;
```

En este caso, el procedimiento mostrar tendrá que implementarse dos veces, debido a que se ha definido como un método virtual. No es factible sin embargo la implementación de métodos con el mismo nombre pero con diferente tipo de argumentos dentro de la misma

[1] Son métodos diferentes aunque con el mismo nombre y son utilizados en las clases heredadas, para que cada una de ellas tenga su propia implementación.

clase (constructores o destructores), lo que constituye una considerable desventaja en relación a otros lenguajes orientados al objeto.

Turbo Pascal no admite la sobrecarga de operadores como parte del polimorfismo, característica que ya ha sido implementada por otros lenguajes.

Turbo Pascal es un gran lenguaje no cabe duda, sin embargo todavía no consigue un funcionamiento óptimo si se requiere de usar adecuadamente los conceptos que sustentan a la programación orientada a objetos. No obstante, cabe esperar con el advenimiento de nuevas versiones, mejoras significativas en este renglón.

TURBO PROLOG

Turbo Prolog es un lenguaje de programación de tipo declarativo, es decir el programador únicamente debe describir el entorno de solución que rodea al sistema, sin especificar la forma en que se accesará dicha solución. Esto se debe a que estos lenguajes llevan consigo un mecanismo de inferencia ya implementado (en el caso de Prolog es un mecanismo de encadenamiento hacia atrás). Además, los lenguajes declarativos tienen un formato específico para la representación del conocimiento, que en PROLOG es en base a reglas de producción.

Una implementación en PROLOG consta de 4 secciones principales: dominios, predicados, claúsulas y objetivos.

Dominios. En esta sección se declara los tipos de datos (symbol, integer, real, listas, etc.) que serán utilizados por las reglas del sistema, asignándoles un identificador a cada uno de ellos. La siguiente declaración de dominios es válida:

```
DOMAINS
   persona = symbol /* se declara un symbol, un tipo de dato */
   colonia = symbol /* similar al string
   lista = real*     /* se declara una lista de reales*/
   edad =integer    /* se declara un entero */
```

Predicados. En esta sección se definen los tipos de argumentos que utilizarán las reglas que conforman a nuestro sistema. Considerando la sección de dominios anterior, la siguiente declaración de predicados es válida:

```
PREDICATES
   registro (persona, edad, colonia)
   mayor (persona,persona)
```

Clausulas. En esta sección el programador desarrolla las reglas y los hechos que conforman a la base de conocimiento de nuestro sistema. Cada regla o hecho debe terminar con un punto, y además si se utilizan variables, éstas deben comenzar con mayúsculas:

```
CLAUSES
  /* H E C H O S */
  registro (carlos,25,polanco).
  registro (pepe, 22, aragon).
```

```
registro (fabiola, 11, irrigacion).
registro (alejandro,19, polanco).

/* R E G L A */
mayor ( P1,P2) if
  registro (P1, Edad1, _) AND
  registro (P2, Edad2, _) AND
  Edad1 > Edad2 AND
  write (P1,' es mayor que ',P2).
```

Obsérvese que la regla definida en el ejemplo anterior define a P1 como mayor que P2 dependiendo de la edad respectiva con que cuenten. Observe la utilización de la variable anónima (_), que a diferencias de las normales no reciben ningún valor debido a que la información esperada no es relevante para el resultado en cuestión.

Objetivos. Esta sección puede ser definida por el programador (objetivo interno) o por el usuario (objetivo externo). Aquí se declara el objetivo que perseguirá el sistema, requisito indispensable para que comienze el funcionamiento del mecanismo de inferencia hacia atrás. Dado el siguiente objetivo:

```
GOAL
  mayor (pepe,alejandro)
```

El resultado será:

```
"pepe es mayor que alejandro"
TRUE
```

También es posible utilizar el mecanismo de inferencia para proporcionar varias soluciones:

```
GOAL
  mayor (carlos, X)

  carlos es mayor que pepe
  X = pepe
  carlos es mayor que alejandro
  X = alejandro
  carlos es mayor que fabiola
  X = fabiola
```

Prolog constituye un lenguaje de vanguardia dentro de los lenguajes de tipo declarativo, ya que además de incorporar una biblioteca de instrucciones muy amplia, cuenta con la capacidad de interrelacionarse con bases de datos implementadas en una gran variedad de paquetes comerciales, así también cuenta con un manejo bastante aceptable de listas y archivos definidos por el usuario.

TURBO C++

En mi punto de vista, Turbo C++ constituye definitivamente el lenguaje más completo de todos los que existen en la actualidad y por consiguiente el ideal para desarrollar cualquier tipo de aplicación, entre ellos Sistemas Expertos.

Turbo C++ es un superconjunto del lenguaje C, y la novedad es la incorporación de rutinas que facilitan la implementación de la programación orientada a objetos. Explicaré algunos detalles para la implementación de sistemas.

La creación de una clase en C++ se lleva a cabo de la siguiente

manera:

```
CLASS vehiculos  {
    int ruedas;
    float peso;
  public:
    int muestra_ruedas (void);
    void asigna ( int r, float p);
} obj1 ;
```

Observe como en la misma declaración de la clase vehiculos se crea el objeto obj1. Turbo C++ necesita que se especifiquen los prototipos de los métodos que conforman a la clase, especificando el tipo de argumentos que recibirán, así como el tipo de dato que retornarán, en caso de funciones.

La descripción de los métodos de la clase se efectúa de la siguiente manera:

```
int vehiculos :: muestra_ruedas (void)
  {
    return ruedas;
  }
void vehiculos :: asigna (int r, float p)
  {
    ruedas = r;
    peso = p;
  }
```

En el programa principal, los métodos pueden ser accesados de la siguiente forma:

```
main () {
...
cout << "De el numero de ruedas y el peso"
cin >> ri;
```

```
cin >> p1;
objeto1.asigna (r1,p1);
cout << "ruedas: " << muestra_ruedas ()
...
}
```

Ahora examinaremos la forma en que Turbo C++ implementa las tres características básicas de todo lenguaje orientado al objeto: encapsulamiento, herencia y polimorfismo.

1) Encapsulamiento. Turbo C++ lo lleva a cabo de una manera muy profesional, pues el acceso a los datos y a los métodos pueden ser de tipo público (public), privado (private) o protegido (protected), según las necesidades del programador. Por omisión el acceso es privado.

Los elementos definidos como públicos pueden ser accesados en cualquier sección del programa fuente, a diferencia de los elementos de tipo privado, en el que su manejo está restringido únicamente al ámbito de los métodos públicos que la conforman. Los elementos de tipo protegido son un tipo especial de elementos privados y únicamente tienen significado en la herencia de clases.

Generalmente la mayoría de los métodos son declarados como públicos para que puedan ser accesados en el programa principal de nuestro sistema, pero el código de datos raramente tendrá que definirse así, aunque Turbo C++ lo permite. En el ejemplo anterior, ruedas y peso fueron definidos como privados y esto origina que la siguiente implementación sea erronea:

```
main () {
...
cout << "De el numero de ruedas y el peso"
cin >> r1;
cin >> p1:
objeto1.asigna (r1,p1);
cout << "ruedas: " << objeto1.ruedas     /* E R R O R */
...
}
```

Turbo C++ permite incluso la utilización de funciones que no son parte de la clase pero que pueden tener acceso a los elementos privados (protected) de ésta. Estas funciones son denominadas friend y son utilizadas principalmente cuando diversas clases requieren de la utilización de una misma función.

2) Herencia. Turbo C++ la lleva a cabo de una forma completa, permitiendo incluso la utilización de herencias múltiples. Un ejemplo es el siguiente:

```
/* clase base.*/
CLASS vehiculos  {
    int ruedas;
    float peso;
  public:
    int muestra_ruedas (void);
    void asigna ( int r, float p);
} ;

/* clase derivada */
CLASS vehic_carga : public vehiculos {
    char tipo_carga [80];
  public:
    char muestra_carga (void);
    void asigna_carga (char *tc);
} obj2;
```

Observe la necesidad que existe de declarar un tipo de acceso en la clase derivada. Esto va a decidir la forma en que los métodos de vehic_carga accesarán a los elementos particulares de la clase base vehiculos.

Cuando el acceso es público, los elementos públicos de la clase base seguirán siéndolo en la clase derivada, y los de tipo privado seguirán siendo privados, lo que imposibilita su manipulación directa, aunque la mayoría de las veces no es necesaria. En el caso de un acceso de tipo private los elementos públicos de la clase base pasarán a ser privados en la derivada, y al igual que en el acceso public los elementos private lo seguirán siendo en la clase derivada.

Turbo C++ permite incluso un acceso denominado protected el cual permite una manipulación de los elementos protected de la clase derivada. Como se había mencionado un elemento de tipo protegido es similar al de tipo privado, excepto que pueden ser manipuladas por métodos friend.

Turbo C++ permite que una clase tome características de más de una clase base, especificando para cada uno de ellos un tipo de acceso específico:

```
CLASS base1 {
   int b11,b12;
 public:
   int proc11 (void);
 }
```

```
CLASS base2 {
  protected:
   int b21,b22;
  public:
   int proc12 (void);
 }

CLASS derivada : public base1 , protected base2 {
  int d11;
  public:
  int procd1 (void);
 } obj_d;
```

El objeto obj_d, el cual es una instancia de la clase derivada tiene los atributos de las clases base1 y base2. De acuerdo al acceso dado, este objeto puede acceder a los datos b21 y b22, y a los procedimientos proc11, proc12, pero no a los elementos b11 y b12, debido a que éstos son de tipo privado.

3) Polimorfismo. La implementación que realiza C++ para el manejo de esta importante característica es excepcionalmente poderosa y a la vez clara. El polimorfismo en Turbo C++ puede llevarse a cabo ya sea mediante la sobrecarga de funciones (métodos normales, constructores, destructores y virtuales), o con la sobrecarga de operadores. La sobrecarga de funciones normales se lleva a cabo como se muestra en el siguiente ejemplo:

```
CLASS vehiculos  {
    int ruedas;
  public:
    int muestra_ruedas (void);
    void asigna ( int r);
    void asigna ( char r);
} objeto;

void vehiculos :: asigna (int r) {
```

```
   ruedas = r;
 }

void vehiculos :: asigna (char *r) {
   ruedas = atoi (r);  /* conversion a numero */
  }
```

Observe la utilización de la instrucción de biblioteca atoi (requiere stdlib.h) para realizar la conversión de una cadena a un entero. Con esto el programador no tiene que preocuparse en el tipo de datos que será pasado al método asigna de la clase vehiculos, pues automáticamente el compilador en tiempo de ejecución realiza la ligadura (late binding). Las siguientes sentencias en el programa principal son perfectamente válidas:

```
objeto.asigna (10);
objeto.asigna ("10");
```

La sobrecarga de funciones también puede llevarse a cabo mediante constructores o destructores. Los constructores son métodos que se ejecutan automáticamente cuando un objeto es creado, mientras que los destructores son métodos que se ejecutan cuando un objeto deja de existir en el sistema. Un método constructor se declara con el mismo nombre de la clase a la que pertenecen al igual que los destructores, excepto que éstos anteponen el operador ~:

```
class coordenadas {
   int x,y;
   public:
      void coordenadas (x1,y1) { x=x1; y=y1}
      void ~coordenadas {printf ("objeto destruído") }
      void muestra_coordenadas;
 }
```

```
main (void) {
  coordenadas coord1 (10,4) , coord2 (3,2);
  return 0;
}
```

En el ejemplo anterior el sistema únicamente ejecuta el método constructor (al crear los objetos) y el método destructor (al salir de la función main). Observe también el uso de argumentos a partir de la declaración del objeto, lo que da a Turbo C++ una gran versatilidad.

El polimorfismo también puede llevarse a cabo en operadores (sobrecarga de operadores). Supongamos que se requiere de sobrecargar los operadores más (+) y de asignación (=) para hacer más eficiente la suma de objetos que representen un conjunto de coordenas cartesianas:

```
CLASS cartesiano {
   int x,y;
  public:
   cartesiano operator+ (cartesiano op2);
   cartesiano operator= (cartesiano op2);
   void muestra (void);
   void asigna (int xx, int yy);
  } a,b,c;

// sobrecargando el operador + //
cartesiano cartesiano :: operator+ (cartesiano op2) {
  cartesiano aux;

  aux.x = x + op2.x ;
  aux.y = y + op2.y ;
  return aux;
 }

// sobrecargando el operador = //
cartesiano cartesiano :: operator= (cartesiano op2) {
```

```
  x = op2.x
  y = op2.y
  return *this;
}
```

Como los signos más (+) e igual (=) son operadores binarios, el operando que manda a llamar a la función definida es el primero, mientras que el segundo operando es pasado como argumento de la función definida. El compilador determina en tiempo de ejecución cual de los operandos habrá de ejecutarse, el procedimiento más estándar o éste, de acuerdo al tipo de dato qua el operador correspondiente reciba como argumento. Obsérvese que el tipo de argumento recibido por los operadores son de la clase cartesiano, así también se especifica en los prototipos de la funciones que una estructura del mismo tipo será dada como resultado.

En la función que define al operador igual (=) se hace mención a this. Este es un puntero que apunta al objeto actual, que en este caso es el que manda a llamar la función (operando de la izquierda). Como el operador más modifica el valor del operando izquierdo es necesario devolver este puntero ya que de otra manera su valor se perdería al salir de la función. Observe como los elementos del lado izquierdo no tienen la necesidad de especificar el prefijo indicador al objeto al que pertenecen:

```
x = op2.x;
```

En este caso x hace referencia al elemento x del operando izquierdo, mientras que op2.x es el elemento x del operando derecho. Si a,b,c son objetos de la clase cartesiano, las

siguientes sentencias son válidas:

```
main () {
  ...
   a.asigna (4,3);
   b.asigna (7,3);
   c = a + b;  // suma de objetos, utilizando el más definido //
   ...
}
```

Turbo C++ cuenta también con herramientas que hacen amigable su utilización, como son el uso de rutinas que permiten la implementación de interfases potentes y fácilmente adaptables a cualquier tipo de hardware con que se cuente. Así también incorpora su propia librería para operaciones de entrada y salida, debido a que con el advenimiento de esta tecnología es necesario tomar en cuenta algunas operaciones que no están definidas en el estándar ANSI de C, como son la entrada y salida en archivos, de unidades de datos definidos por el usuario.

Sin duda, Turbo C++ es el lenguaje cuyo uso ha crecido debido a su potencia para implementar en forma explícita la programación orientada a objetos. Este lenguaje constituye una herramienta esperada por todo programador profesional, debido a que tiene el control de prácticamente todos los detalles que deben considerarse en sistemas que se precien de ser eficientes. No cabía esperar menos, ya que Turbo C++ es un trabajo de muchos años de los mejores programadores de compiladores en los Estados Unidos.

CAPITULO VIII

DESARROLLO DE UNA APLICACION DE SISTEMAS EXPERTOS

En este capítulo realizaré una pequeña implementación de un Sistema Experto aplicado a la selección de personal de la empresa X S.A de C.V.

EL objetivo no es desarrollar la metodología para desarrollar Sistemas Expertos sino visualizar la forma en que se realiza la implementación de aplicaciones en una herramienta comercial.

ANALISIS

La empresa X S.A. de C.V., dedicada al reclutamiento y selección de personal altamente calificado, requiere de un Sistema Experto que auxilie al ejecutivo de cuenta en la elección de personal viable para ocupar el puesto de programador.

El trámite que se sigue en la selección del personal es el siguiente:

1) El aspirante llena una solicitud, donde tiene que proporcionar datos como su nombre, edad, estudios, gustos, vicios, enfermedades, experiencia, conocimientos que posee dentro dentro del puesto, y el sueldo mensual que desea.

2) La recepcionista de la empresa asigna a un ejecutivo de cuenta en particular el trámite del aspirante.

3) El ejecutivo de cuenta seleccionado entrevista al aspirante para conocer la forma en que se desenvuelve, y observar su presentación, además verifica que la solicitud haya sido llenada fielmente.

4) El ejecutivo de cuenta posteriormente revisa las características que debe reunir el aspirante para considerarlo como viable.

5) El aspirante viable realiza un examen, calificado por un experto en el ramo.

6) Si el aspirante tiene una calificación aprobatoria, se verifica que la plaza esté aún libre, de lo contrario los datos de la persona son almacenados en una base de datos de personal en espera de ser contratado.

7) Se informa al aspirante de la situación en la que se encuentra, y se procede a los trámites necesarios en caso de aceptación.

El sistema a desarrollar se enfocará básicamente en el punto 4, y será usado por el ejeucutivo de cuenta o por un auxiliar del mismo, para conocer la viabilidad del aspirante.

Los requisitos con que debe contar el personal que desea el puesto de programador son los siguientes:

1) Tener un 1 año de experiencia en el puesto como mínimo comprobable y tener entre 24 y 35 años.

2) Ser titulado de las carreras de ingeniería en computación o licenciado en Informática o ingeniero en sistemas computacionales, y tener un promedio mínimo de 8.

3) No fumar, ni beber y contar con buena presentación.

4) Tener conocimientos en Clipper, Pascal y bases de datos.

5) No pedir un sueldo superior a los 5'000,000.

SOLUCION DEL PROBLEMA

Como hemos visto, un candidato viable al puesto de programador requiere de cumplir con cinco condiciones:

```
SI objetivo1=cumplido Y objetivo2=cumplido Y...
                          objetivo5=cumplido
ENTONCES candidato=viable
SINO     candidato=no_viable
```

Para cada objetivo establecido es necesario realizar una regla. Por ejemplo para objetivo1:

```
SI Experiencia >= 1 Y Edad >=24 Y Edad <=35
ENTONCES Objetivo1=cumplido
SINO     Objetivo1=no_cumplido
```

De igual forma se desarrolla el conjunto de reglas para los demás objetivos.

VP EXPERT

Para la implementación del problema utilizaré el entorno de desarrollo Vp Expert de Paperback Software Internacional, debido a

su excelente legibilidad y además a la incorporación de un lenguaje de programación muy accesible y poderoso. Explicaré algunas de las instrucciones con que cuenta:

ACTIONS marca el comienzo del bloque de acciones de Vp-Expert. Está sección describe el objetivo del sistema.

ASK Pregunta al usuario por el contenido de las variables, si éstas son necesitadas en la búsqueda de objetivos.

BECAUSE Se especifica la razón de la regla en cuestión.

BKCOLOR=n Define el color del fondo de la pantalla, representado por n (0=negro, 1=rojo, etc.).

CHOICES var : opcion1, opcion2,..; Especifica el rango de valores que puede asignarse a la variable en cuestión.

CLS Limpia la pantalla.

COLOR=n Define el color con que se desplegará la información.

DISPLAY "mensaje" Despliega en el monitor un mensaje. Las variables deben ir entre corchetes ({}).

FIND objetivo Fuerza la búsqueda del objetivo señalado en la base

de conocimiento.

IF condicion THEN accion1 **ELSE** accion2 Estructura tradicional utilizada en la base de conocimiento para la implementación de reglas. Si la condicion se cumple el bloque de accion1 se ejecuta, de lo contrario el bloque de accion2 se lleva a cabo.

RULE nombre Marca el comienzo de una nueva regla, especificando el nombre de la misma.

UNKNOWN Constante definida en Vp Expert y asumida por aquellas variables a las que todavía no se les asigna un valor.

! Comentario.

El objetivo de un programa Vp Expert es aquel dado en la sección de **ACTIONS**. Las instrucciones son ejecutadas secuencialmente y únicamente cuando se encuentra la instrucción **FIND** se realiza la consulta a la base de conocimiento mediante un mecanismo de encadenamiento hacia atrás. Algunas de las condiciones de las distintas reglas permiten que el control del programa se transfiera a la sección de variables, donde el usuario proporcionara los valores que éstas deben contener. La estructura de un programa Vp Expert es el siguiente:

<instrucciones de configuración>
ACTIONS

```
  <conjunto de instrucciones>
    ....
RULE n1
  IF <condicion>
  THEN accion1
  ELSE accion2;
...
...
ASK var1 : "Accese la variable 1: "
CHOICES var1 : valor1, valor2,....
...
...
```

Las instrucciones de configuración son utilizadas para afectar los valores por defecto del paquete, como puede ser el color del fondo de la pantalla.

El bloque de acciones (ACTIONS) en nuestro problema deberá desplegar una pantalla que indique el problema a resolver por el sistema. Así también es necesario que ejecute un objetivo para accesar el nombre del usuario y su puesto, y por último buscar en la base de conocimiento el objetivo principal: la viabilidad del aspirante.

El bloque de reglas (Rules) deberá definir la regla principal a cumplir, además de cada uno de los subobjetivos que la conforman.

El bloque de preguntas (ASK-CHOICES) deberá de describir a todas aquellas variables cuyo valor podría ser consultado al usuario en caso de que el sistema lo requiera. Por ejemplo el nombre del aspirante, su edad, su experiencia, etc.

El bloque de acciones tendrá básicamente la estructura

siguiente:

```
ACTIONS
<presentación del sistema>
FIND subobj01
FIND candidato
<presentación de respuestas, declarando el estado de cada objetivo>
```

El bloque de reglas estará conformado como sigue:

```
RULE 1
IF nombre<>a AND puesto=programador
THEN subobj01=cumplido
ELSE subobj01=no_cumplido
     obj1=no_es_el_puesto
     obj2=no_es_el_puesto
     obj3=no_es_el_puesto
     obj4=no_es_el_puesto
     obj5=no_es_el_puesto;

RULE 1
IF subobj01=cumplido AND obj1=cumplido AND obj2=cumplido AND ...
THEN candidato=viable
ELSE candidato=no_viable;

RULE 2
IF experiencia >= 1 AND edad >=24 AND edad <=35
THEN obj1=cumplido
ELSE obj1=no_cumplido
     obj2=categoria_1_fallo
     obj3=categoria_1_fallo
     obj4=categoria_1_fallo
     obj5=categoria_1_fallo;
...
...
```

En caso de que un subobjetivo falle, es necesario asignar a los subojetivos posteriores una variable de referencia de que no han sido aplicados, debido a que no se cumplió un objetivo anterior. Lo anterior hace posible una pequeña explicación al usuario del

sistema.

La implementación en la sección de preguntas y respuestas será de la forma que sigue:

```
ASK experiencia : "De la experiencia en años del aspirante..";
...
ASK  fuma : "El aspirante fuma ?";
CHOICES fuma : Si,No,Casi_nunca,No_se_sabe;
...
```

El sistema completo es el siguiente:

```
! PROGRAMO: Amílcar Amado Monterrosa Escobar
! ABSTRACTO: Prototipo de Sistema Experto aplicado a la selección
!            de personal
! FECHA     : 15 de enero de 1992

BKCOLOR=3;

ACTIONS

COLOR=15
DISPLAY"
                              X S.A. DE C.V."

COLOR=0
DISPLAY "
     Prototipo de Sistema Experto que determina si una persona es
     un candidato viable para ocupar el puesto de programador

"
COLOR=9
DISPLAY "                              Presione cualquier tecla

     -"
CLS
COLOR=15

CLS
! Se buscará en la base de conocimiento un objetivo que
! defina a SUBOBJ01 para accesar el nombre del candidato
! y verificar si el puesto requerido es el de programador
```

```
FIND subobj01

! Se busca en la base de conocimiento el objetivo CANDIDATO
! el cual define la viabilidad del aspirante

FIND candidato

! A continuación se despliegan los resultados obtenidos:

CLS
DISPLAY " El analisis efectuado da como resultado:

   1. Un ano de experiencia como minimo y 24 a 35 anos
                              ====> {obj1}
   2. Titulado en computacion y promedio superior a 8
                              ====> {obj2}
   3. No fuma, ni bebe y cuenta con buena presentacion
                              ====> {obj3}
   4. Conocimientos en Clipper, Pascal y bases de datos
                              ====> {obj4}
   5. Sueldo no mayor a 5'000,000
                              ====> {obj5}


         {Nombre}, es un candidato {Candidato} para
          el puesto de programador
"
COLOR=9
DISPLAY "                          < Presione cualquier tecla >

 ~"
COLOR=0
CLS
DISPLAY "

                    Fin de consulta!

 ~";

!************************* COMIENZO DE REGLAS ****************


RULE subobj01
IF   nombre<>a AND puesto=programador
THEN
     subobj01=cumplido
ELSE
     subobj01=no_cumplido
     obj1=no_es_el_puesto
```

```
     obj2=no_es_el_puesto
     obj3=no_es_el_puesto
     obj4=no_es_el_puesto
     obj5=no_es_el_puesto
! Si la regla no se cumple, se asigna a los subobjetivos
! posteriores una referencia que auxilie al usuario en
! el reporte de resultados
CLS;


RULE candidato
! Para que el candidato sea viable debe cumplir todos los objetivos
IF   subobj01=cumplido AND obj1=cumplido AND obj2=cumplido AND
     obj3=cumplido AND obj4=cumplido AND obj5=cumplido
THEN candidato=viable
ELSE candidato=no_viable;


RULE Obj1
IF experiencia >= 1 AND edad >= 24 AND edad <=35
THEN obj1=cumplido
ELSE
! Si el objetivo 1 no se cumple, los demas objetivos tampoco lo
haran:
     obj1=no_cumplido
     obj2=categoria_1_fallo
     obj3=categoria_1_fallo
     obj4=categoria_1_fallo
     obj5=categoria_1_fallo;


RULE Obj2
IF titulo=si AND promedio > 8
THEN obj2=cumplido
ELSE obj2=no_cumplido
     obj3=categoria_2_fallo
     obj4=categoria_2_fallo
     obj5=categoria_2_fallo;


RULE Obj3
IF fuma=no AND bebe=no AND presentacion=bp OR presentacion=mbp
THEN obj3=cumplido
ELSE obj3=no_cumplido
     obj4=categoria_3_fallo
     obj5=categoria_3_fallo;

RULE Obj4
IF clip_pas_bd=si
THEN obj4=cumplido
ELSE obj4=no_cumplido
     obj5=categoria_4_fallo;
```

```
RULE Obj5
IF sueldo <= 5000000
THEN obj5=cumplido
ELSE obj5=no_cumplido;

! ****************** SECCION DE VARIABLES**********************


ASK   Nombre : "El nombre del aspirante por favor...

 - Con apellido paterno y materno.

";

ASK   Puesto :" El puesto solicitado es:
";
CHOICES Puesto: programador, otros, ninguno;


ASK   Experiencia : "La experiencia del aspirante en el puesto es:
";

ASK   Edad : " La edad del aspirante es :";

ASK   Titulo : " El aspirante es egresado de las siguientes
                 carreras:

           - Ingeniero en computación
           - Ingeniero en sistemas
           - Licenciado en informática

";
CHOICES Titulo: Si,No,Tal_vez;

ASK   Promedio : " Su promedio en la Universidad fue:";

ASK  Fuma: "El aspirante fuma ?";
CHOICES Fuma: Si,No, Casi_nunca, No_se_sabe;

ASK  Bebe : " El aspirante bebe demasiado ?";
CHOICES Bebe : Si, No, Tal_vez;
```

```
ASK  Presentacion : "La presentacion del aspirante es

     mp= mala presentacion
     rp= regular presentacion
     bp= buena presentacion
     mbp= muy buena presentacion
";
CHOICES Presentacion :mp,rp,bp,mbp;

ASK Clip_pas_bd : "El aspirante tiene conocimientos de clipper,
pascal y bases de datos ";
CHOICES  Clip_pas_bd : Si, No, Tal_vez;

ASK Sueldo : "El sueldo minimo requerido por el aspirante es ";

! FIN DEL PROGRAMA
```

La edición de este programa puede realizarse en cualquier editor profesional que maneje el formato de texto ASCII, la única recomendación es añadir la extensión KBS al nombre del programa. Vp Expert también cuenta con su propio editor.

Para consultar al sistema anterior es necesario ejecutar Vp Expert de la siguiente manera:

```
C:\VP-EXP> vpx
```

A continuación aparecerá un conjunto de opciones disponibles:

[F1] Help Proporciona la ayuda al usuario en el entorno Vp Expert.

[F2] Induce Crea una base de conocimiento a partir de una hoja de

cálculo, una base de datos o un archivo tipo texto.

[F3] Edit Entra al editor de Vp Expert para modificar el programa actual.

[F4] Consult Consulta la base de conocimiento actual.

[F5] Tree Despliega un conjunto de opciones para visualizar los resultados generados mediante una consulta paso a paso (trace).

[F6] Filename Carga una base de conocimiento especifica.

[F7] Path Cambia la trayectoria actual. Se utiliza cuando las bases de conocimiento están contenidas en otros subdirectorios o en una unidad de disco diferente.

[F8] Quit Salir a DOS.

Para iniciar la consulta del sistema realizado es necesario cargar la base de conocimiento, presionando la tecla [F6] (FILENAME) proporcionando el nombre del archivo físico (en este caso use el nombre x-sacv.kbs). Posteriormente presionamos [F4] para empezar la consulta del sistema:

El nombre del aspirante por favor...

- Con apellido paterno y materno.

-Patricia Monterrosa E.

El puesto solicitado es:

programador otros ninguno

La experiencia del aspirante en el puesto es:

- 1

La edad del aspirante es :

- 26

El aspirante es egresado de la carrera de ingeniero en computacion, ingeniero en sistemas o lic. en informatica ?

Si No Tal vez

Su promedio en la Universidad fue:

- 8.8

El aspirante fuma ?

Si No Casi nunca No se sabe

el aspirante bebe demasiado ?

S: No Tal vez

La presentacion del aspirante es

mp= mala presentacion

rp= regular presentacion

bp= buena presentacion

mbp= muy buena presentacion

mp rp bp mbp

El aspirante tiene conocimientos de clipper, pascal y bases de datos:

Si No Tal_vez

El sueldo minimo requerido por el aspirante es

4500000

Las respuestas anteriores llevarán al siguiente resultado.

El analisis efectuado da como resultado:

1. Un ano de experiencia como minimo y 24 a 35 anos

====> cumplido

2. Titulado en computacion y promedio superior a 8

```
                    ====> cumplido
3. De fina, si bebe y cuenta con buena presentacion
                    ====> cumplido
4. Conocimientos en Clipper, Pascal y bases de datos
                    ====> cumplido
5. Sueldo no mayor a 5'000,000
                    ====> cumplido

   Fabiola Monterrosa I. es un candidato viable para
   el puesto de programador
```

El sistema anterior está conformado de apenas siete reglas, sin embargo es un sistema que permite auxiliar al ejecutivo de cuenta. Las ventajas que habíamos visto en capítulos anteriores con respecto al uso de herramientas comerciales son obvias, el ahorro de código fuente y la facilidad para instrumentarlo. Existen herramientas aún más poderosas que evitan la necesidad de editar las partes que conforman a un Sistema Experto, mediante el empleo de menús.

Un ejemplo interesante sería desarrollar este ejemplo en algún lenguaje de programación como C++ o PROLOG, para tener un conocimiento más detallado de la dificultad a las que nos enfrentaríamos al tener que diseñar el motor de inferencia, la representación del conocimiento y las interfases de comunicación.

La aplicación de sistemas expertos en varias disciplinas es cada vez más factible debido a la aparición de herramientas poderosas que facilitan su implementación. Sin embargo, no debemos olvidar las grandes ventajas que nos proporcionan los lenguajes de programación en el desarrollo de Sistemas Expertos más específicos y flexibles principalmente.

CAPITULO IX

TENDENCIAS

En un futuro no muy lejano se contempla el uso de Sistemas Expertos comerciales aplicados a una amplia gama de actividades dentro de la industria; así también, existe una tendencia a una mejora significativa de los motores de inferencia actuales y a una interrelación con redes neuronales.

APLICACIONES COMERCIALES

El creciente interés por incorporar el uso de Sistemas Expertos en diversas actividades es debido a la facilidad de éstos de auxiliar en la toma de decisiones, actividad fundamental en toda

empresa, además de la pequeña inversión inicial que se realiza.

Es necesario enfatizar que algunas investigaciones desarrolladas en Estados Unidos muestran a los Sistemas Expertos como un auxiliar inteligente en la toma de decisiones y no como un reemplazo de expertos.

Algunas compañías innovadoras, están aprovechando la tecnología de sistemas expertos para crear una base de conocimiento de tipo institucional, es decir sistemas donde el conocimiento de expertos se acumule para permitir a las nuevas generaciones un aprendizaje más rápido y concreto sobre las estrategias de solución anteriormente empleadas. Entre estas compañías, podemos mencionar a General Electric, Boeing, Campbell's, Texas Instruments Canon, Fujitsu, IBM y Northrup.

Algunas ramas de la industria donde ya existe un crecimiento de Sistemas Expertos aplicados a actividades específicas son:

a) Mercadotecnia. Existe una gran variedad de sistemas comerciales aplicados a esta rama, en donde destaca Sales Edge de Human Edge Software, un programa que ayuda al vendedor a desarrollar estrategias de venta para clientes específicos. El sistema interroga al vendedor sobre características específicas sobre la personalidad de ambos, y da información sobre la forma en que debe presentarse el agente de ventas ante el cliente y además

proporciona algunas características generales deseables en el producto.

Actualmente la tendencia es el desarrollo de Sistemas Expertos que auxilien en cuestiones como precios, promoción, producción y distribución de productos.

b) Finanzas. Actualmente circula en forma comercial un Sistema Experto denominado MARBLE, que evalúa aplicaciones de préstamo comerciales, combinando la experiencia de reconocidos profesionales en la materia con principios probabilísticos.

Actualmente la tendencia es el desarrollo de Sistemas Expertos que auxilien al profesional en análisis de crédito y préstamos, establecimiento de políticas para flujo de efectivo, análisis de cartera y pronóstico de ventas.

c) Contabilidad. Uno de los sistemas expertos comerciales más famosos es precisamente ExperTax, una gran herramienta para el planeamiento de impuestos.

Actualmente se desarrollan sistemas expertos con énfasis en estudios de depreciación, compra-venta, e inversión, así como análisis de presupuestos.

d) Manejo de recursos humanos. Es una de las aplicaciones más crecientes en la actualidad, y principalmente se enfoca en el

reclutamiento y selección de personal.

e) Protección de Información. Aunque en la actualidad son muy escasas las aplicaciones para protección de la información en equipos en equipos de cómputo, se espera que resulte un gran apoyo en la lucha contra "virus" informáticos sobre todo.

MOTOR DE INFERENCIA

Actualmente la tecnología que apoya a los dispositivos de la Inteligencia Artificial se dirige al desarrollo de motores de inferencia más sofisticados, destacando los estudios realizados en base a la semántica de Tableau.

Como se ha visto en capítulos anteriores los motores de inferencia se basan principalmente en el método de resolución binaria, en donde los resultados se consiguen mediante la utilización de silogismos[1] disyuntivos (or) o conjutivos (and) de orden racional. Por ejemplo, consideremos una representación para el siguiente enunciado: "Toda persona es hombre o mujer":

mujer (X) or hombre (X). (1)

[1] Silogismo: Argumento que consta de tres proposiciones: la mayor, la menor y la conclusión, deducida la última de la primera por medio de la segunda.

Y la siguiente negación:

-mujer (juan) (2)

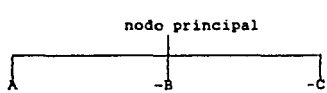
Usando un silogismo disyuntivo que indica "Si cualquier componente de una disyunción está negado, entonces es cierto que la posible solución únicamente se encuentra en el resto de los componentes de la regla". Para este caso, el resto de los componentes se reduce a una sentencia únicamente, por lo que es posible afirmar:

hombre (juan) (3)

El último refinamiento en motores de inferencia está basado en la semántica de Tableau, o también llamado "árbol de la verdad", que permite una utilización más flexible en el manejo de sentencias con lógicas de primer orden principalmente. Supongamos la siguiente disyunción:

A or -B or -C

Que da lugar al siguiente 'arbol':

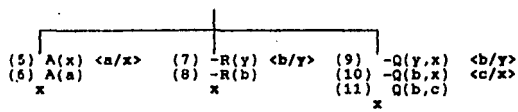


En este caso, para que la fórmula sea cierta, es necesaria que

al menos una sub-fórmula de alguna rama sea verdadera. El árbol resultante además debe de mostrar todos los posibles valores que pueden tener los argumentos de los predicados para permitir su evaluación y con ello llegar a una solución. Consideremos la siguiente disyunción y los elementos dados:

```
(1) A(x) or -R(y) or -Q(y,x)
(2) -A(a)
(3) Q(b,c)
(4) R(b)
----------------------------
```

Entonces el árbol mostrado es el siguiente:



Donde:

x,y son variables

a,b,c son constantes

A,R,Q son predicados

El motor de inferencia de Tableau analiza cada rama del "árbol" generado y analiza aquellos posibles valores que pueden cumplirse. Puede observarse que en todas las ramas no se llega a una solución satisfactoria (x), debido a que no se encuentran en la base de conocimiento los predicados con los argumentos definidos.

La semántica de Tableau constituye una gran herramienta para

facilitar la implementación de algoritmos de búsqueda más sencillos, más directos y por lo tanto más eficientes, principalmente en el manejo de sentencias con lógicas de primer orden. Esta herramienta será sin duda uno de los componentes principales de los lenguajes de programación futuros.

SISTEMAS EXPERTOS-REDES NEURONALES

Actualmente se efectúan pruebas que ayuden a la formación de sistemas que obtengan las mejores características de los Sistemas Expertos con Redes Neuronales, otra de las ramas con mayor auge de la Inteligencia Artificial, ya que se trata de mecanismos verdaderamente "pensantes" que pueden generar sus propias reglas a partir de nuevas situaciones presentadas.

[1]"Una red neuronal es un sistema de computación que consiste de un número simple de elementos que procesan algunos valores de entrada para generar una salida. Las redes unicamente requieren ejemplos específicos de valores de entrada con sus correspondientes valores de salida para poder generar reglas genéricas".

Como su nombre lo indica las redes neuronales simulan el comportamiento de las neuronas que conforman al cerebro humano. Según especialistas en la materia (Rolston, Lawrence, etc.) las

[1] Charles W. Engel

neuronas del cerebro no son tan rápidas como los mecanismos de computadora, sin embargo utilizan un procesamiento paralelo de información que permite generar respuestas en tiempos increíbles.

La principal ventaja que obtienen los Sistemas Expertos de esta unión es la capacidad de generar nuevas reglas a partir de nuevos hechos presentados, constituyéndose en sistemas capaces de "aprender".

La difusión de estos sistemas será considerable con el advenimiento de computadoras con un procesamiento paralelo de información.

CONCLUSIONES Y DIRECTRICES PARA TRABAJOS POSTERIORES

Sin lugar a dudas los Sistemas Expertos constituyen actualmente una parte muy significativa en el ámbito de la comunidad informática, y el creciente interés por mejorar las técnicas actuales hacen prever que todavía hay mucho por hacer con relación a estos sistemas.

Dentro de la elementos que conforman a los Sistemas Expertos es importante realizar mejoras en el motor de inferencia para un manejo más eficiente de mecanismos con lógicas de primer orden,

cuestión que aparentemente empieza a ser atendida mediante la innovación de técnicas más completas, como la de Tableaus.

En lo que se refiere a las representaciones de conocimiento, aunque los sistemas basados en reglas de producción son las más usadas, los objetos reunen características muy importantes que permiten su consideración para desarrollar Sistemas Expertos: legibilidad, flexibilidad y una gama muy amplia de lenguajes de programación que permiten su implementación directa. Con el advenimiento del lenguaje C++, el diseño de Sistemas Expertos en base a objetos será, sin duda, el más popular.

Como anteriormente se mencionó, una forma de solucionar los procesos de aprendizaje de los Sistemas Expertos es mediante la union con redes neuronales, y aunque ciertamente involucra un costo considerable, en ocasiones algunas aplicaciones lo requieren. Esta interfase debe ser estudiada en forma más profunda, ya que llevada a cabo en forma conveniente servirá como base para el desarrollo de sistemas que se asemejarán aún más al comportamiento humano.

Otra cuestión muy importante es el de las interfases de comunición, donde es necesario aprovechar las técnicas que brindan otros dispositivos de inteligencia artificial como pueden ser el procesamiento de lenguaje natural y los sistemas de visión por ordenador.

La Inteligencia Artificial avanza a pasos agigantados en la busca de técnicas auxiliares y en un futuro no muy lejano la "inteligencia" será un rasgo de todos los sistemas y entonces está ciencia pasará a ser el elemento principal de los sistemas informáticos.

INDICE DE TABLAS Y FIGURAS

Capitulo 3:

Capitulo 6:

BIBLIOGRAFIA

1. David W. Rolston
Principles of Artificial Intelligence
Mc Graw Hill

2. Brian Sawyer
VP-Expert, Rule-Based Expert System Development Tool
Paperback Software

3. Dieter Nebendahl
Sistemas Expertos. Introducción a la técnica y aplicación
Marcombo

4. Richard Fairley
Software Engineering Concepts
Mc Graw Hill

5. Philip R. Robinson
Using Turbo Prolog
Mc Graw Hill

6. Herbert Schildt
Using Turbo C++
Mc Graw Hill

7. Herbert Schildt
Artificial Intelligence using C
Mc Graw Hill

8. Herbert Schildt
Advanced Turbo Prolog
Mc Graw Hill

9. Juan Pablo Sanchez y Beltran
Sistemas Expertos: una metodología de programación
Macrobit

10. John E. Hopcroft and Jeffrey D. Ullman
Introduction to automata theory, languajes and computation
Addison-Wesley Series in Computer Science

11. Browston, Farrell, Kant and Martin
Programming Expert System in OPS5
Addison-Wesley

12. T. Budd
A little Smalltalk
Addison-Wesley

13. E. Charniak and D. McDermont
Introduction to Artificial Intelligence
Addison-Wesley

14. B. Cox
Object-Oriented Programming: An Evolutionary Approach
Addison-Wesley

15. M. Burke and L. Genise
Logo and Models of Computation
Addison-Wesley

16. L. Moskowitz
Ruled-Based Programming
Byte, 11 (12), 217-224

17. L. Brodie
Starting Forth
Prentice-Hall

18. G. Steele
Common Lisp
Digital Press

19. Joseph Schmuller
Languages of Artificial Intelligence
PC AI, September/October 1991, 21-25

20. Joseph Schmuller and Yingsheng Mi
Expert System Shells At work (Mentor)
PC AI, September/October 1991, 42-47

21. Hal Berghel
Logic Programming with Tableaus
PC AI, September/October 1991, 48-50

22. Merrill E. Warkentin
Artificial Intelligence in Business and Management
PC AI, November/December 1991, 26-28

23. Charles W. Engel and Margaret Cran
Pattern Classification (neural networks)
PC AI, May/June 1990, 20-28

24. Joseph Schmuller and Yingsheng Mi
Expert System Shell at work (VP Expert)
PC AI, May/June 1990, 52-56

25. R. Burke, A. Rangaswamy and J. Eliashberg
A Knowledge Based System for Advertising Design
Marketing Science, 9(3), 212-229

26. Thomas C. Bartee
Expert Systems and AI, Aplications and Management
Howard W. Sams & Company