"INVESTIGACION SOBRE LA FILOSOFIA Y ASPECTOS TECNICOS DE LENGUAJES DE PROGRAMACION REFERENTES A SISTEMAS EXPERTOS Y DISEÑO DE UN PROTOTIPO ORIENTADO AL AREA DE ORTOPEDIA"

TRABAJO DE GRADUACION
PREPARADO PARA LA FACULTAD
DE INGENIERIA

PARA OPTAR AL GRADO DE:

INGENIERO EN CIENCIAS DE LA COMPUTACION

POR

BLANCA LIDIA ALAS CASTRO # 9204100
JOSE EDUARDO PLEITEZ TECUN # 9504024

MARZO-1999

SOYAPANGO-EL SALVADOR-CENTROAMERICA

UNIVERSIDAD DON BOSCO

RECTOR

ING. FEDERICO HUGUET RIVERA

SECRETARIO GENERAL

PBRO. PEDRO JOSE GARCIA CASTRO

DECANO DE LA FACULTAD DE INGENIERIA

ING. CARLOS GUILLERMO BRAN

ASESOR DE TRABAJO DE GRADUACION

LIC. REINA DE ALVARADO

JURADO EXAMINADOR

ING. ANA MERCEDES CACERES

LIC. FRANCISCO ANTONIO DIAZ

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

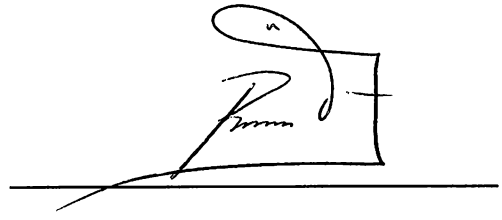
DEPARTAMENTO DE INGENIERIA EN COMPUTACION

JURADO EVALUADOR DEL TRABAJO DE GRADUACION

"INVESTIGACION SOBRE LA FILOSOFIA Y ASPECTOS TECNICOS DE LENGUAJES DE PROGRAMACION REFERENTES A SISTEMAS EXPERTOS Y DISEÑO DE UN PROTOTIPO ORIENTADO AL AREA DE ORTOPEDIA"

ING. ANA MERCEDES CACERES

LIC. FRANCISCO DIAZ



LIC. REINA DE ALVARADO

RECONOCIMIENTOS

Las personas que a continuación se mencionan proporcionaron su aporte técnico para la adquisición del conocimiento en el área de ortopedia:

- Dra. Canizalez, encargada de atender a los pacientes del ISRI designada por la GTZ.
- Ing. Carlos Matios, profesor de la Universidad Don Bosco en el área de ortopedia y técnico protesista del proyecto GTZ.
- Ing. Henner encargado del proyecto GTZ.

Con la ayuda o el aporte información que estas personas brindaron fue posible la realización del prototipo, que analiza y diagnóstica prótesis para el miembro inferior.

AGRADECIMENTOS

En primer lugar, doy gracias a Dios, por haberme dado la oportunidad de vivir, guiarme en mi camino, por permitirme lograr una de mis metas ser INGENIERO, la cual ha sido posible con la iluminación y sabiduría que tu me haz brindado.

A la virgen por haber sido guía y luz en mi camino.

A mis padres: Margarito y Eva por haberme dado la vida, por educarme y guiarme por el buen camino; También les doy las gracias por todos los sacrificios realizados, que Dios los bendiga por siempre.

A mis hermanas: Marta y Marianita por haber confiado en mí y darme su apoyo.
A mi hermana: Ana María que esta en el cielo por ser mi Angelito.

A una familia especial en mi vida: Tía Hilda, José Alberto, Sandra, Manuel, Marielos por darme un espacio en su hogar, porque con el apoyo de ellos se hizo posible terminar una de mis metas, les agradezco mucho por todo lo que han hecho por mí, los quiero mucho que Dios los bendiga.

A mis Abuelos: Virginia, Manuel, Laura, Pedro por el apoyo que me han brindado, los quiero mucho que Dios los proteja.

A mis Tías: Blanca Rubia y familia, Dominga y familia, Clelian y familia, Ofe y familia, por haberme apoyado, desde la distancia que se encuentran, que Dios las bendiga siempre.

A mis seres queridos: Mariana, Manuel, Oscar que ya no se encuentran en la tierra, pero que formaron parte de mi educación.

A Don Ricardo, Karlita, Lidia, Hector, Candy, Ricardo, Rafael, Valeria, por haberme brindado su apoyo moral, y brindado su confianza que Dios los bendiga.

A mis amigas: especialmente Karlita por que siempre me ha brindado su apoyo, comprensión en todo momento, a Sandra, Claudia, Leyla, Karina, Xiomara, Pauly, Yessenia por su constante apoyo e interés que Dios las bendiga.

A mi amigo Carlos Quintanilla, aunque ya no se encuentre en este mundo terrenal, se que me ha brindado su apoyo.

A José, quien fue mi compañero de tesis, le doy gracias por haberme permitido trabajar juntos, y por haber sido un verdadero compañero.

A Lic. Reina de Alvarado: por haber sido una excelente asesora, por su constante empeño brindado.

A mis jurados: Ing. Cáceres, Lic. Diaz por habernos apoyado.

A los que fueron mis profesores: por compartir sus conocimientos.

A todas las personas, compañeros, que brindaron su apoyo e información de manera desinteresada.

Blanca Lidia Alas.

Doy gracias a Dios por haberme permitido lograr una más de mis metas, por haberme iluminado el camino para llegar a este momento tan importante .

Le doy gracias a mis padres por haberme traído al mundo, haber confiado en mi, haberme brindado todo su amor y apoyo y haberme dado la oportunidad de realizar mis estudio. De igual forma le agradezco a mi hermana por su cariño y apoyo, y por todo esto y mucho más dedico les dedico este gran triunfo.

Agradezco de todo corazón a Mirna por la paciencia que tuvo al esperarme todo este tiempo, por haber estado conmigo en las buenas y las malas, por amarme tanto todos estos años y haberse sacrificado conmigo en algunas ocasiones, por todo esto y muchas cosas más le dedico especialmente este triunfo.

Agradezco infinitamente a Francisco por su amistad, confianza y por haberme brindado todos sus conocimientos sobre el tema que sin ningún recelo compartió conmigo para enriquecer los míos y así haber podido llegar al final de este camino.

Al mono (Nilson) y al pato (Sergio) por su amistad, el apoyo y la confianza que tienen en mí.

Les doy las gracias a todos mis compañeros de la Super: Einar, Elisa, Karina, Quique, Gaby, Omar, Fran, Claudia, Ligia, Juan Francisco, Cecy, Angel, Jenny, Milton, Ethel y Cristhy por haber confiado en mi y haberme brindado todo su apoyo.

A José Mario, Paty, Ana Daysi, Erika por su amistad y confianza en mi.

A Juan Francisco Cardona Conde y Ricardo Herrera por toda la ayuda que me brindaron durante el tiempo que estudiamos juntos en la universidad, por su apoyo y sobre todo por su gran amistad. Gracias amigos.

A mis suegros don Luis y Vicky y mis cuñados Adrian y Rebeca por su confianza y preocupación.

Finalmente agradezco a todos mis profesores, amigos y compañeros que brindaron su apoyo de manera desinteresada.

José Eduardo Pleitez Tecún.

PROLOGO

El presente documento constituye el trabajo de graduación denominado "Investigación sobre la Filosofía y aspectos Técnicos de lenguajes de Programación referentes a Sistemas Expertos y Diseño de un Prototipo Orientado al área de Ortopedia".

En un inicio se presenta una introducción sobre el trabajo a realizar, luego se presentan aspectos teóricos sobre Inteligencia Artificial, Sistemas Expertos sobre los cuales se dan a conocer su definición, el área donde son aplicados, y la forma de adquisición del conocimiento.

A continuación se presentan aspectos generales que deben poseer los lenguajes de programación para Inteligencia Artificial, luego se dan a conocer las metodologías de la investigación que fueron utilizadas.

Enseguida se presenta las diferentes técnicas utilizadas, después se presenta la metodología más adecuada para la realización de un Sistema Experto, así mismo se muestra la guía para interactuar con el experto y finalmente se presenta el análisis y diseño del prototipo.

INDICE

CONTENIDO **PAGINAS**

INTRODUCCION

La Inteligencia Artificial estudia cómo lograr que las máquinas realicen tareas que, por el momento, son realizadas mejor por los seres humanos, y es una rama de la ciencia de la computación que se puede definir como el estudio sistemático del comportamiento inteligente, con el fin de imitar o simular las habilidades humanas mediante la utilización de máquinas y computadoras. Los Sistemas Expertos son uno de los puntos que componen las investigaciones en el campo de la Inteligencia Artificial. Un sistema de computadoras que trabaje con técnicas de Inteligencia Artificial deberá estar en situación de combinar información de forma "inteligente", alcanzar conclusiones y justificarlas. Los Sistemas Expertos son una expresión de los sistemas basados en el conocimiento.

Un Sistema Experto es un programa computacional capaz de resolver problemas específicos con una habilidad equiparable a la de los expertos humanos, representan una nueva oportunidad a la computación, abren sendas que hasta la fecha estaban cerradas a aplicaciones específicas y permiten abordar nuevos problemas; además trabajan con una base de conocimiento[1], en la cual se almacenan premisas relativas a una cierta área del conocimiento y con un motor inferencial[2], que es el cerebro del Sistema Experto, encargado de razonar con la información almacenada en la base del conocimiento.

Para el desarrollo de Sistemas Expertos existen lenguajes de programación especializados tales como: LISP, PROLOG, OPS5, VISUAL BASIC, VISUAL FOX, VISUAL C++ y muchos otros que facilitan la representación y manipulación del conocimiento humano.

[1] (bc) Base de Conocimiento.

El propósito de este documento es proporcionar a la persona sin conocimientos previos en el área de Inteligencia Artificial, los conceptos básicos para comprender lo que son los Sistemas Expertos, presentar algunos lenguajes de programación que existen en el medio cuya aplicación es el área de Sistemas Expertos, y presentar el diseño de un prototipo orientado a una aplicación específica del área de ortopedia.

[2] (mi) Motor inferencial.

OBJETIVOS

1.1 Objetivo general

- Realizar una investigación acerca de los diferentes lenguajes para la creación de Sistemas Expertos y la metodología que se debe seguir para la creación de un prototipo el cual será aplicado al área de ortopedia.

1.2 Objetivos específicos

- Investigar cual ha sido la evolución histórica de los Sistemas Expertos desde sus orígenes.
- Investigar acerca de los lenguajes de Inteligencia Artificial PROLOG, LISP, Ops5, Visual Basic, Visual fox, Visual C++.
- Investigar la filosofía y los aspectos técnicos de los lenguajes antes mencionados.
- Presentar el diseño de un prototipo aplicando las técnicas y sintaxis de uno de los lenguajes investigado.
- Explicar la forma en que se debe interactuar con un experto para obtener el conocimiento en el área específica de aplicación para que ayude al programador a crear la base del conocimiento.
- Explicar cuales son los pasos a seguir para crear un programa experto.
- Diseñar un prototipo experto orientado al área de ortopedia.

ALCANCES Y LIMITACIONES

2.1 ALCANCES

- Investigación de la filosofía y los aspectos técnicos de los lenguajes Prolog, Lisp, Ops5, Visual Basic, Visual Fox, Visual C++.
- Aplicar los conocimientos de programación adquiridos a lo largo de la carrera.
- Explicar los pasos a seguir para la creación de un prototipo experto.
- El prototipo contendrá la información requerida o necesaria para la solución de un problema específico.
- Investigación de una técnica de la Inteligencia Artificial, que será la de los Sistemas Expertos.
- Presentar las condiciones que deben existir para aplicarse un Sistema Experto.
- El trabajo presenta cuales son los distintos lenguajes utilizados en sistemas Expertos, sus características y ventajas.

2.2 LIMITACIONES

Una de las características de los Sistemas Expertos es que se basan esencialmente en el conocimiento del experto, por lo que el trabajo se limita:

- A la información que el experto considere apropiada para resolver un problema en el campo de actividad específica.
- La portabilidad del prototipo dependerá de la versión del software (Shell) que se utilizará para programar.
- El diseño del prototipo será únicamente de un área específica de Ortopedia.

CAPITULO I
ANTECEDENTES

1.1 Situación actual

Actualmente los sistemas expertos representan una nueva oportunidad en aplicaciones específicas para resolver problemas complejos, ya que es en estas áreas donde tiene que funcionar un sistema experto.

La Inteligencia Artificial es una presencia aparente de experiencias y conocimientos. Se trata de técnicas de solución que son el resultado de la investigación del comportamiento humano. En la actualidad los Sistemas Expertos son una técnica eficaz de solución de problemas, su comunicación con el usuario posee una interfaz amigable, lo que permite que el usuario lo opere con facilidad sin tener mayores conocimientos en el área de computación.

En El Salvador los Sistemas Expertos son una de las tecnologías menos utilizadas, ya que el desconocimiento por parte de los posibles usuarios es evidente, como pudo constatarse en las encuestas realizadas a profesionales y estudiantes de diferentes carreras.

Estas encuestas arrojaron como resultado que un 20 por ciento de las personas encuestadas han escuchado o tienen conocimientos de alguna aplicación de los Sistemas Expertos en un área específica, por ejemplo se mencionaron áreas como:

- área médica: diagnóstico y prescripción de medicamentos
- controlar el tráfico inteligente
- simulación de diseño de mapas
- medidor de caudales de ríos para abrir compuertas de presas

- fallas de equipo
- robótica en industria automotriz

De este 20 por ciento que tienen conocimientos sobre los Sistemas Expertos, todos son graduados en alguna carrera, principalmente en computación, pero con maestrías en el extranjero que es de donde han adquirido ese conocimiento.

El resto de los encuestados son estudiantes, los cuales no tienen ningún conocimiento de aplicaciones de los Sistemas Expertos aunque algunos conozcan de la existencia de ellos.

Los resultados de la encuesta reflejaron también que la mayoría de las personas desconocen que los Sistemas Expertos son una de las técnicas de la Inteligencia Artificial, la cual presenta grandes oportunidades en cualquiera de las aplicaciones donde supone una descarga del conocimiento del experto en un trabajo rutinario.

En la actualidad los pocos Sistemas Expertos que existen en El Salvador están hechos a un nivel de tésis, por ejemplo se puede mencionar el Sistema Experto para la orientación vocacional creado para el sistema de admisiones de la Universidad José Simeón Cañas (UCA). Los ejemplos mencionados anteriormente son ejemplos claros de que la Inteligencia Artificial ha sido poco explorada en el país, esto se debe a que la información que pueda obtenerse es reducida. Unicamente se conocen un par de tesis de la facultad de ingeniería de la Universidad José Simeón Cañas, un par de libros de la biblioteca de la Universidad Don Bosco y algunos cursos que se han organizado sobre el tema, que sirven de apoyo a la institución educativa en sus labores y como apoyo bibliográfico al público en general.

CAPITULO II
JUSTIFICACION

Uno de los beneficios que se obtendrán de la investigación, será mostrar que la técnica de los Sistemas Expertos es útil para la solución de problemas de la vida real; no sólo de los países desarrollados sino también en El Salvador. Para esto la población debe tener conocimiento de qué es un Sistema Experto, cuáles son los requerimientos de desarrollo, qué Software se puede utilizar para crear una aplicación orientada a cualquier área donde exista un problema complejo con comportamiento dinámico o explosión combinatoria, donde no resulte posible o rentable una solución convencional de procesamiento de datos.

Otro factor de peso es conocer los aspectos de mayor importancia que debe cumplir un lenguaje de programación para Inteligencia Artificial; ya que la función primaria de estos lenguajes es la representación del conocimiento del experto a través de métodos formales de representación, entre estos métodos se encuentran la lógica de predicados.

La metodología a presentar tiene como objetivo desarrollar los pasos a seguir para construir un programa de Inteligencia Artificial, en cualquier área de aplicación, debido a que en todo programa se tiene que desarrollar una secuencia lógica a seguir.

Estos son los principales beneficios que se desean adquirir u obtener a largo del proyecto de investigación.

CAPITULO III
MARCO TEORICO

Inteligencia Artificial

Como una introducción a lo que es el desarrollo de Sistemas Expertos es necesario conocer los orígenes de la Inteligencia Artificial y algunos conceptos fundamentales.

En los apartados de este capítulo se presenta historia de cómo se inició la Inteligencia Artificial, definiciones, así como sus áreas de investigación, subcampos en que ésta es dividida, también se mencionan sistemas basados en la inteligencia, e inteligencia computacional.

3.1 Historia.

En el siglo XVIII Wolfgang Von Kemmpelen construyó un autómata jugador de ajedrez que había confundido y asombrado por muchos años a Europa, pero con el tiempo se demostró que se trataba de un fraude porque ocultaba en su interior ingeniosamente a un jugador humano, en esta época no existía la tecnología necesaria para crear una máquina inteligente.

No fue hasta el año de 1950 cuando Alan Turing público el artículo llamado "Can a machine think?" (¿Puede una máquina pensar?), pero mientras que su interés en éste era el de un lógico matemático, el de los científicos en computación era mucho más práctico. Al verse que tan fácil podrían utilizarse las computadoras para resolver los problemas que le resultaban demasiado tediosos a los seres humanos, comenzó a especularse si también pudiesen usarse para resolver aquellos que las personas encontraban en extremo difíciles.

John McCarthy fue la persona que en 1955 uso el término Inteligencia Artificial para englobar todas las actividades encaminadas a la construcción de sistema inteligentes, aunque él mismo ha opinado que sería mejor utilizar el término Inteligencia Mecánica debido a la mala interpretación que puede hacerse de su significado. McCarthy opina que la finalidad de ésta, es resolver problemas que requieren inteligencia, pero sin obligación de utilizar los mismos mecanismos.

A partir de la reunión denominada "Darmouth Summer Resecart Project with Artificial Intelligence" que McCarthy convocó en 1956 en el Darmout College, se configura la Inteligencia Artificial como una de las ramas de la computación con su vida propia, aglutinando un gran número de actividades (robótica, comprensión del lenguaje natural, visión artificial, aprendizaje, programación automática, razonamiento, planificación, resolución de problemas) y con dos filosofías distintas.

Una de las corrientes filosóficas es la de McCathy y Minsky en el MIT (Massachussets Institute of Tecnology, Instituto de Tecnología de Massachussets) en la cual pretenden la construcción de

máquinas inteligentes, esto significa, sistemas cuyo comportamiento sea como si lo llevase a cabo una persona experta.

Y la de Newell y Simon en la Carnigie Melon University, dedicados a estudiar modelos de comportamiento humano para construir sistemas inteligentes por emulación del cerebro humano, incluso en su estructura.

En 1956 surgieron los primeros programas de Inteligencia Artificial creando programas que jugaban ajedrez y damas, los cuales demostraban teoremas de lógica y geometría, resolviendo integrales y aprendiendo conceptos.

El primer logro significativo llego en 1957 cuando Newell, Shaw y Simon crearon un programa llamado GPS (siglas de General Problem Solver, que significa resolvedor general de problemas), con el cual trataron de resolver problemas, aunque no los abarcaron todos.

Fue entonces cuando los investigadores de la Inteligencia Artificial añadieron un segundo objetivo a su investigación: descubrir como funciona la mente humana.

3.2 Definición.

Existen diferentes definiciones de Inteligencia Artificial entre las cuales se mencionan las siguientes:

"La Inteligencia Artificial define las técnicas de la lógica formal, de los nuevos procedimientos y métodos de búsqueda de la base de la representación del conocimiento en programas de ordenador, por otro lado, se define como la presencia aparente de experiencia y conocimiento de causa en programas de ordenador"[3].

"Es el estudio de cómo hacer que los ordenadores hagan cosas que, en estos momentos hace mejor el hombre"[3].

[3] Dieter Nebendahl, Sistemas Expertos, Barcelona, Maracombo Boixareu Editores, 1987, cap. 1

"Es la capacidad de un ser vivo o de una máquina de ordenar informaciones externas, observaciones, experiencias, descubrir interrelaciones, valorarlas con las informaciones para abstraer de esta forma cosas y poderlas ligar entre sí"[3].

"Estudia como lograr que las máquinas realicen tareas que, por el momento, son realizadas mejor por los seres humanos"[4].

3.3 El área de investigación de la Inteligencia Artificial.

Aún cuando la Inteligencia Artificial puede ser usada en diversidad de campos, en la actualidad las áreas donde más se ha utilizado es en la psicología, filosofía, informática, y la lingüística.

3.4 Subcampos de la Inteligencia Artificial.

La Inteligencia Artificial se divide en varios subcampos entre los cuales se mencionan los siguientes:

Sistemas de lenguaje natural.

El sistema de procesamiento de lenguaje natural, es aquel en que parte de la información a procesar está codificada en lenguaje natural y se aplican algoritmos para el análisis sintáctico, y pragmático de la información.

Entre las tareas de lenguaje natural de la vida diaria se encuentran: comprensión, generación y traducción.

Sistemas reconocedores de imágenes.

[4] Elaine Rich. Kevin Knight, Inteligencia Artificial, Segunda Edición, España, Mc Graw-Hill, 1994, pág 3

En los reconocedores de imágenes debe identificarse el significado de imágenes con ayuda de procesos exactamente definidos. El tratamiento de imágenes abarca todos los procesos de cálculo para la manipulación de datos aportados por los mismos, en especial, solo de aquellos que tratan una imagen para obtener otra, por ejemplo: filtrado, asilado, aumento del contraste, correcciones geométricas y eliminación de errores.

Robótica.

Los robots han ocupado siempre la fantasía de muchas personas. Sin embargo, los robots de las modernas plantas industriales de producción desempeñan actividades monótonas y repetitivas según un plan de acción previamente fijado con exactitud. Con las técnicas de Inteligencia Artificial se intenta que el comportamiento de los robots sea más inteligente.

Sistemas Expertos.

Los Sistemas Expertos buscan almacenar el conocimiento de expertos para un campo de especialidad determinada y estrechamente delimitada, y así solucionar un problema mediante la deducción lógica. Representan la transición del procesamiento de datos al procesamiento de conocimientos y sustituyen al mismo tiempo los algoritmos por mecanismos de inferencia.

Los Sistemas Expertos encuentran aplicaciones donde haya conocimientos especializados y experiencias, y no resulte posible o rentable una solución convencional de procesamiento de datos.

Estos sistemas utilizan métodos de representación y modelación del conocimiento en problemas de búsqueda en espacios de estados, métodos de inferencia y automatización del razonamiento, demostración automática de teoremas en lógicas de orden superior y métodos formales para verificación, síntesis y transformación.

Se trabaja sobre el impacto de estos métodos de representación y razonamiento en el diseño de agentes inteligentes.

CAPITULO IV
ESTUDIOS SOBRE SISTEMAS EXPERTOS

Sistemas expertos

La evolución histórica de Sistemas Expertos, conceptos básicos, campos de aplicación, sus componentes esenciales son tratados en este apartado.

Se presenta además una ampliación sobre la base de conocimientos, el mecanismo de inferencia, el componente explicativo, la interfase de usuario, el componente de adquisición, y del equipo o las personas que participan en el desarrollo del Sistema Experto.

4.1 Historia sobre sistemas expertos.

Los Sistemas Expertos son una rama de la Inteligencia Artificial, nacieron en los laboratorios de Universidad de Stanford en la década de 1960 para ayudar a diagnosticar infecciones en la sangre; desde entonces los Sistemas Expertos se han movido virtualmente dentro de cada profesión que requiere de juicio humano.

En 1964 Joshua Lederberg, profesor de genética en la Universidad de Stanford, diseñó un programa para enumerar las configuraciones posibles, pero válidas de un determinado conjunto de átomos. Denominó a su programa DENDRAL, que es una abreviatura de DENDRitic Algoritmo (algoritmo dendrítico).

El proyecto DENDRAL tuvo por objetivo formular hipótesis acerca de la estructura molecular de un compuesto. Hoy en día DENDRAL no es solo un programa sino una familia de programas donde el algoritmo original esta en el centro de la familia y los demás programas ampliaron significativamente su poder. La importancia del DENDRAL creció con gran rapidez, de tal manera que en 1983 se construyó una compañía independiente para su construcción y mejoramiento. DENDRAL buscó soluciones en la dirección contraria a la que siguieron todos los demás sistemas, es decir que este proyecto buscaba métodos específicos dependientes del campo de actividad. Este cambio de dirección fue bautizado por el profesor Feigenbaum como "cambio de paradigma de Inteligencia Artificial" donde se sustituyeron las técnicas basadas en el poder por otras basadas en el conocimiento, estableciendo las bases de los sistemas expertos.

En la década de 1970 se desarrolló el sistema experto llamado MICIN, cuyo propósito era ayudar al médico, quien tenia 4 decisiones por tomar: si el paciente sufre de alguna infección bacteriana, qué organismo es el causante, qué fármacos podrían ser adecuados, cuáles de ellos administrar.

La forma de trabajo es la siguiente: con base a los datos del paciente y a los resultados del análisis, llega a una conclusión para cada una de las cuatro preguntas. Posteriormente nació PROSPECTOR, el cual fue diseñado para ayudar a los geólogos en la búsqueda de depósitos de minerales y en la evaluación del potencial mineralógico de grandes zonas geográficas.

Estos sistemas fueron considerados como los modelos intelectuales de los sistemas expertos.

En el área de ortopedia se han diseñado algunos sistemas expertos, entre los que podemos mencionar:

- Sistema de computadora interactivo para almacenaje, búsqueda de datos clínicos estandarizados, y material asociado con los implantes ortopédicos, desarrollado en Septiembre de 1981 en la ciudad de COOK SD, este sistema fue diseñado por IRAP y se construyó para ser ocupado por personal sin entrenamiento en computación.
- Sistema para producir una intraoperatibilidad personalizada de prótesis del fémur desde medidas tomadas durante un procedimiento quirúrgico, desarrollado en diciembre de 1989, el cual fue elaborado por personas pertenecientes al departamento de cirugía ortopédica de la Universidad Katholieke, en Leuven, Bélgica. Sistema que fue desarrollado para permitir al cirujano crear una prótesis en cualquier cavidad femoral lo que permite un implante mientras la operación esta en progreso utilizando una computadora técnica mecánica.
- CAPP (Computer Assisted Preoperative Planning), es un sistema experto utilizado en la cirugía ortopédica para el planeamiento del tratamiento quirúrgico, creado en junio de 1990 en el hospital ortopédico del colegio médico de la Universidad de Zegreb en Yugoslavia.
- CAD-CAM, desarrollado en 1992 para el reemplazo total de la cadera. Este sistema fue alimentado con un conocimiento básico que se refería al factor de el proceso de diseño. Era un programa de computadora que había sido provisto con reglas de diseño pre-programadas. Este

Sistema Experto contribuyó a personalizar las implantaciones ortopédicas de cadera hechas en el departamento de Ing. Biomédica del Instituto Ortopédico, en Stanmore en Middlesex.

- El 1994 fueron desarrollados dos planes de cirugía para la compleja reconstrucción en un caso de cirugía ortopédica. El objetivo era comparar la ventaja y limitaciones del método común utilizando radiografías bidimencionales, comparadas con el uso del software llamado MCAE (Mechanical Computer Assisted Engineering) utilizando rayos "X", y datos de una topografía computarizada.
- CAD-CAM, desarrollado en 1997 y es utilizado en Alemania para los implantes de Titanium para la reconstrucción del defecto craneal y craneofacial.

Se han publicado varios artículos sobre uso de los sistemas expertos en el área ortopédica, a continuación se presentan los siguientes:

- En Octubre de 1986 es publicado un artículo sobre simulación ortopédica mediante microcomputadoras el cual fue hecho por Samuels H. Este artículo trata sobre el significado de adaptar material gráfico para mejorar los esfuerzos de la educación ortopédica utilizando el microcomputador.
- En 1995 Diener P., Burgun A., Cleret M., Le Beaux P., pertenecientes al departamento de informática médica de la Universidad Hospital Pountchaaillou, en Francia publicaron un artículo titulado "La Representación del Conocimiento en Ortopedia", desde el modelo conceptual hasta el nomenfactural utilizando Orthanay. Este sistema es una construcción basada en conocimiento para la codificación de los registros en los pacientes.

4.2 Definición.

Un Sistema Experto se entiende como un nuevo tipo de software que imita el comportamiento de un experto humano en la solución de un problema. Pueden almacenar conocimientos de expertos para un campo determinado y solucionar un problema mediante deducción lógica de conclusiones.

4.3 Campos de aplicación.

La aplicación de Sistemas Expertos es adecuada donde los expertos disponen de conocimientos complejos en un área estrechamente delimitada, donde no existan algoritmos elaborados o donde los existentes no puedan solucionar algunos problemas y no existan teorías completas.

La siguiente tabla nos da una visión de los campos de aplicación en diferentes sectores:

Aplicación de los Sistemas Expertos según los sectores[5]:

Sector / Aplicación	Banca Seguros	Industria	Comercio Servicios	Cargos Estatales
Control de procesos de supervisón	-Observación de tendencias	-Control de procesos. -Gobierno de procesos. - Aviso de estados de excepción.	- Observación de tendencias.	- Control de centrales nucleares o de grandes redes (agua, gas)
Diseño		-Configuración. -Instalaciones fabriles. -Diseño de productos.	- Requisitos de productos.	- Redes de distribución (correos, energía)
Diagnóstico	-Concesión de créditos. -Comprobación de hipotecas. -Análisis de siniestros.	-Motivo de fallo. -Mantenimiento.	-concesión de créditos. -Cálculo de riesgos.	-Diagnóstico médico (hospitales). -Diagnóstico técnico. -Economía energ energética.
Planificación	-Análisis de riesgos. -Gestión de valores. - Planificación de inversiones.	-Funciones lógicas de proyectos. - Proyectos.	-Análisis de riesgos. -Análisis del mercado.	-Planificación de inversiones. -Planificación de emergencias. Planificación de la distribución.

[5] Dieter Nebendahl, Sistemas Expertos, Barcelona, Maracombo Boixareu Editores, 1987, pág. 30

Sector / Aplicación	Banca Seguros	Industria	Comercio Servicios	Cargos Estatales
Asesoramiento	-Asesoramiento de clientes.	-Asesoramiento de clientes.	-Asesoramiento de clientes. -Servicios especiales.	-Asesoramiento de clientes.
Formación	-Formación de colaboradores. -Formación de servicio exterior.	-Formación de colaboradores.	-Formación de colaboradores. -Formación del servicio exterior.	-Formación interna en cuestiones jurídicas.

4.4 Componentes de los sistemas expertos.

Se presentan a continuación los componentes esenciales del Sistema Experto:

La Base de Conocimientos:

Esta contiene el conocimiento de los hechos y de las experiencias de los expertos en un dominio determinado.

Mecanismo de Inferencia:

Permite simular la estrategia de solución de un experto.

Componente Explicativo:

Explica al usuario la estrategia de solución encontrada y el porqué de las decisiones tomadas.

La Interfase de Usuario:

Sirve para que éste pueda realizar una consulta en un lenguaje lo más natural posible.

Componente de Adquisición:

Ofrece ayuda a la estructuración e implementación del conocimiento en la base de conocimientos.

4.5 Equipo de desarrollo de un sistema.

Las personas que participan en el desarrollo de un Sistema Experto desempeñan tres papeles distintos los cuales se mencionan a continuación:

El experto:

Esta es la persona que pone sus conocimientos especializados a disposición del Sistema Experto.

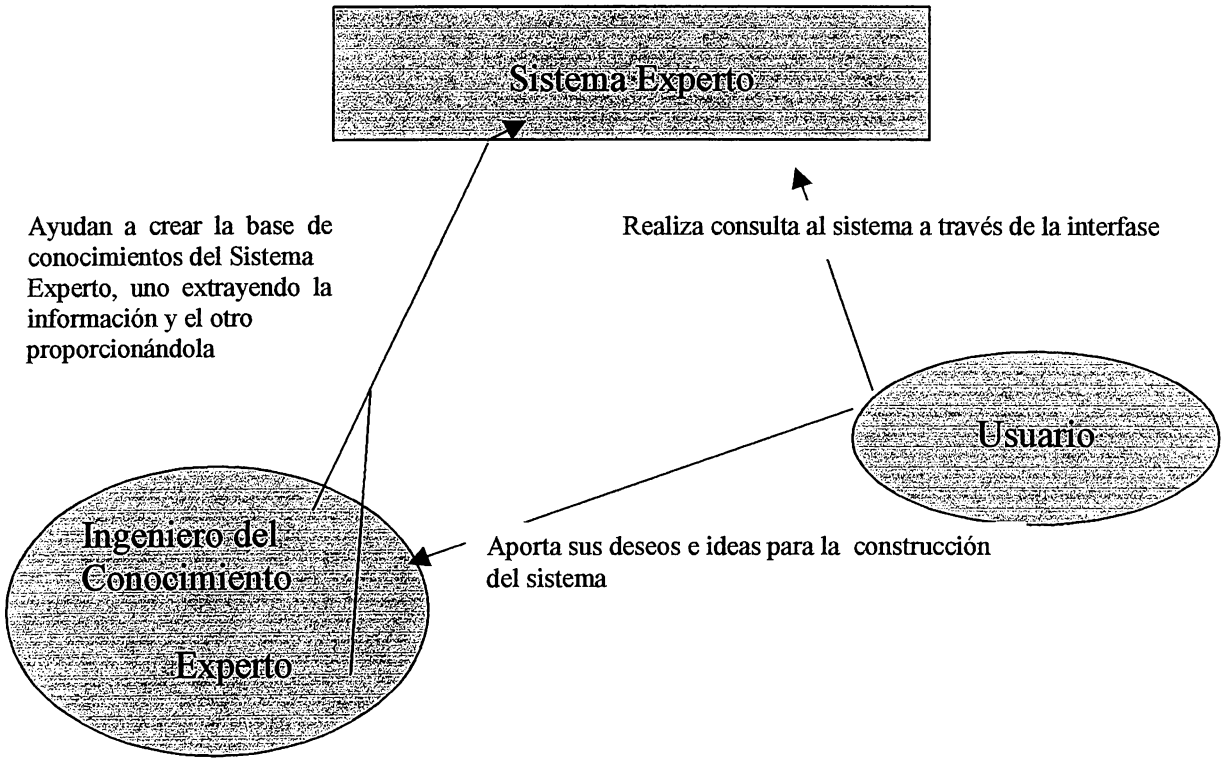
El ingeniero del conocimiento:

Es la persona que plantea preguntas al experto, estructura sus conocimientos y los implementa en la base de conocimientos.

El usuario:

Es la persona que aporta sus deseos y sus ideas, determinando especialmente el escenario en el que debe aplicarse el Sistema Experto. El usuario debe disponer de un conocimiento especializado lo suficientemente amplio sobre el entorno en el cual trabaja para poder manejar el Sistema Expertos.

RELACION ENTRE EL EQUIPO DE DESARROLLO DE UN SISTEMA EXPERTO[6]



[6] Dieter Nebendahl, Sistemas Expertos, Barcelona, Maracombo Boixareu Editores, 1987, pág. 40

4.6 Adquisición y representación del conocimiento

Considerando los sistemas expertos desde la perspectiva del ingeniero de sistemas, se pueden enfocar dos problemas específicos que presentan, los cuales surgen del hecho de que estos sistemas incorporan conocimientos. Uno de ellos es el problema de cómo obtener este conocimiento, o sea el problema de la adquisición del conocimiento; el otro es como codificar y almacenar este conocimiento, o sea el problema de la representación del mismo.

La adquisición y representación del conocimiento es un proceso global que depende del tipo o nivel de conocimiento que se tenga y sobre la base de esto se determinan las técnicas y métodos a utilizar para la adquisición y representación de este conocimiento.

Para el desarrollo del sistema es necesario identificar los tipos o niveles de conocimientos que existen para determinar que técnica o método utilizar.

4.6.1 Tipos de conocimientos.

Existen cuatro tipos o niveles de conocimientos: estructurante, heurístico, epistemológico y el conceptual.

4.6.1.1 Nivel Estructural

En este nivel se representa el "esquema" mental del experto y contiene básicamente la clasificación de las principales entidades que intervienen en el dominio del problema, dichas entidades pueden ser objetos, acciones, situaciones, y otros. De igual forma este tipo de conocimiento posee básicamente las relaciones entre estas entidades.

Aquí se conforma la base del conocimiento de cualquier experto y define el mecanismo que genera la certeza y varia de acuerdo con el campo de experiencia. Así, por ejemplo, en mecánica o geología, las evidencias no son del mismo orden, los procedimientos de valoración son distintos y las vías de acceso a la certeza varían.

Con este tipo de conocimiento un experto toma una cierta actitud ante un problema cuyos pasos lógicos se podrían describir de la siguiente forma: en primer lugar el experto se percata de que le llega mucha información de la cual realiza un análisis y concluye dos cosas, una es que no toda esa información es útil y la otra es que puede faltar información relevante y en último lugar se da cuenta que no necesita prestar mucha atención a cierta clase de información. Esta actitud lo lleva a resolver el problema de una forma rápida y generando buenos resultados.

4.6.1.2 Nivel heurístico

La solución completa de un problema requiere en muchos casos del análisis exhaustivo de todas las posibilidades hasta encontrar la solución. Este proceso es llamado "explosión combinatoria" y lleva mucho tiempo, es por eso que el experto utiliza heurísticas adquiridas a partir de la propia experiencia del experto dando la solución de los problemas que le conciernen.

Asépticamente una heurística sería cualquier cosa que sirva para reducir el espacio de búsqueda, es muy intuitiva y su calidad depende de la habilidad de los expertos para identificar factores relevantes ponderándolos con su importancia relativa e interrelacionandolos adecuadamente y es por eso que se constituyen como la característica principal del conocimiento del experto. Es sano aclarar que no necesariamente una heurística conduce a una solución óptima.

Una heurística debe ser sencilla y fácil de explicar, además debe poseer una probabilidad alta de solución próxima al optimo y una posibilidad baja de llegar a una solución pobre.

El conocimiento en este nivel se corresponde a procedimientos rutinarios ligados a técnicas estándar o de trabajo, muchos de los cuales no están en manuales y constituyen una parte importantísima de la experiencia, así como a reglas practicas utilizadas por el experto en la realización de su tarea.

4.6.1.3 Nivel Epistemológico

Este nivel viene condicionado (o se deriva) por las cuatro I (Incertidumbre, Incompletitud, Inconsistencia e Impresisión), que aparecen en todo el conocimiento que, naturalmente, se utiliza en entornos del mundo real.

La Incertidumbre se produce porque los datos, en los que se basan los razonamientos son inciertos es decir, son verdaderos o falsos, pero no hay forma de saberlo, o porque las reglas de inferencia se obtienen directamente de la experiencia o son heurísticas y, por lo tanto, no completamente fiables. La forma de tratar la Incertidumbre es a través de factores o coeficiente de certidumbre.

La Incompletitud se presenta debido a que no todo lo pertinente al caso en curso puede observarse o recopilarse, o porque no existen suficientes recursos para efectuar todas las deducciones potencialmente interesantes, o porque aun las teorías del mundo solo son aproximaciones. Esto implica la exigencia de razonar en base a suposiciones. La forma de tratar la Incompletitud es asociando respuestas por omisión a ciertas cuestiones.

La Inconsistencia surge, en parte, por el manejo de la Incompletitud, por ejemplo, al pasar por alto una omisión, lo que provoca una contradicción, y, en parte, porque la información puede proceder de fuentes contradictorias; y, también, porque a veces el razonamiento es no-monótono. La forma de tratar la Inconsistencia es a través del mantenimiento de una red de dependencias entre los hechos que luego pueden emplearse para restablecer la consistencia y a través del razonamiento desde distintos puntos de vista.

Las expresiones son imprecisas, borrosas, vagas o, aun, difusas, si en ellas no se define exactamente al menos alguna de las variables que engloban. Algunas de las expresiones que presentan estas característica son aquellas cuantificadoras tales como: algo, mucho, la mayoría, y otros. La forma de tratar la imprecisión es a través de la lógica difusa.

4.6.1.4 Nivel conceptual.

Estos modelos conceptuales pueden verse como una descripción abstracta de cómo un experto resuelve un problema y los tipos de conocimientos necesarios para resolver estos. En efecto, el conocimiento acerca del mundo real implica relaciones causales, temporales y espaciales; y si es acerca de las personas, que se mueven y actúan en este mundo, implica creencias, motivaciones, planes, etc.

La conceptualización no es fácil porque los modelos científicos tienen una cantidad ilimitada de conocimientos demasiados finos. Lo que implica desarrollar teorías, de sentido común o cualitativas, tanto para el entorno como para las personas.

Además debería ser adecuada para que sirva de base a los procesos de razonamiento y su conocimiento asociado, puestos en practica por el experto, esto implica explosión combinatoria.

También representa el modelo del mundo real, del dominio global de aplicación que tiene el experto. En este tipo de conocimiento intervienen cuestiones espaciales y temporales aparentemente fuera del problema.

4.6.2 Adquisición del conocimiento

La adquisición de conocimientos es el proceso de conformar el contenido de conocimientos de los sistemas expertos, la denominada base de conocimientos. Se trata de un proceso que se lleva a cabo a través de toda la duración del sistema. Abarca varias tareas de: obtener conocimientos, organizarlos, codificarlos, validarlos y ponerlos a punto.

Durante la obtención del conocimiento se deben efectuar varias operaciones de las cuales las más importantes son :

- Extraer el conocimiento exteriorizándolo, de tal manera que permanezca disponible para su inspección y manipulación.
- Volverlo explícito acumulando suficientes detalles para hacerlo claro y darle plena expresión.
- Registrarlo de manera simbólica.
- Verificarlo comparando la forma simbólica con el enunciado y la intención originales.

La obtención proporciona elementos aislados de conocimiento que deben organizarse en un todo unificado. La cantidad de organización necesaria depende muchísimo de la forma en que se obtenga y utilice la base de conocimientos.

4.6.2.1 Técnicas y métodos de adquisición del conocimiento.

Diferentes niveles de conocimientos implican diferentes técnicas y métodos de adquisición del conocimiento.

En este sentido se listan a continuación estas técnicas y métodos por tipo o nivel de conocimiento:

Técnicas en el nivel estructurante:

En este nivel de conocimiento son utilizadas las entrevistas, las cuales no deben ser estructuradas, con el propósito de identificar a los expertos, ver la idoneidad de la tecnología, establecer necesidades de los usuarios, familiarizar al ingeniero del conocimiento con el dominio, producir información para la base de conocimiento y ayudar al diseño del sistema. Estas entrevistas deben de tratar sobre tareas que son conocidas, para familiarizar al ingeniero del conocimiento con el dominio del problema y producir información para la primera base de conocimiento. Además deben de tratar de obtener información limitada y acerca de restricciones de procesos con el propósito de determinar las estrategias de razonamiento y heurísticas.

También se utilizan técnicas automáticas tales como: la técnica de la inducción y la del cluster.

Técnicas para el nivel heurístico:

Al igual que en el nivel estructurante, en este se utilizan entrevistas no estructuradas que buscan extraer información limitada y sobre restricciones de proceso; la diferencia es que estas tratan sobre tareas especiales y además buscan extraer información de "casos correosos", es decir flexibles y elásticos.

Técnicas para el nivel epistemológico:

En este nivel son utilizadas únicamente las entrevistas estructuradas para la obtención de conocimientos.

Técnicas para el nivel conceptual:

Este tipo de conocimiento es obtenible como consecuencia de la integración de los distintos tipos de conocimiento. Se intenta en la ultima fase del desarrollo de un sistema experto y el método básico es el de entrevistas estructuradas, claramente asociado al proceso de validación y refinamiento.

La estructura conceptual se hace común y esto debe conseguirse en base a la equivalencia de los formalismos de representación del conocimiento que veremos en la siguiente sección.

4.6.3 Representación del conocimiento

Para el procesamiento y la manipulación del conocimiento, en Sistemas Expertos es necesario formalizar y estructurar dicho conocimiento. En su mayor parte, se dispone del conocimiento a través de entrevistas con los expertos en forma de descripciones de casos o en partes de su actividad.

Los métodos formales de representación del conocimiento son distintos aspectos de la lógica; por ejemplo, lógica de predicados, lógica modal, lógica multivaluada y lógica difusa.

Estos métodos formales y matemáticos no son, sin embargo, imprescindiblemente métodos auxiliares apropiados para comunicarse con expertos de los más diversos sectores especializados.

Por ello se han desarrollado procedimientos de representación del conocimiento que pueden ofrecer un apoyo eficiente a la estructuración y al procesamiento del saber.

Se trata aquí de:

- Reglas de producción: que se basan en la lógica de predicados, una descripción del saber en forma de reglas "sí ....., entonces ...".
- Redes semánticas: que son una representación gráfica del saber sobre objetos y sus relaciones.
- Frames (marcos): son estructuras de datos para la representación de objetos.
- Factores de certeza: los cuales son factores arbitrarios de valoración que suelen encontrarse casi siempre dentro de los limites de –1 a +1. Por ejemplo podría determinarse que el – 1 podría ser "seguro que no", el – 0.5 "probablemente no", el 0 "desconocido", el + 0.5 "probablemente si" y el + 1 "seguro que sí". El ámbito de valores dentro de solución elegido es continuo.

La base de estos procedimientos, sea en la representación o en el procesamiento del conocimiento, es en cierta manera el cálculo de predicados, que es la deducción lógica de resultados que mediante el cumplimiento de determinadas condiciones puede extraerse una deducción lógica. La solución puede tener el valor de "verdadero" o "falso".

4.6.3.1 Formalismos de representación del conocimiento

Según el nivel de conocimiento se adecuan unos formalismos mas que otros, así:

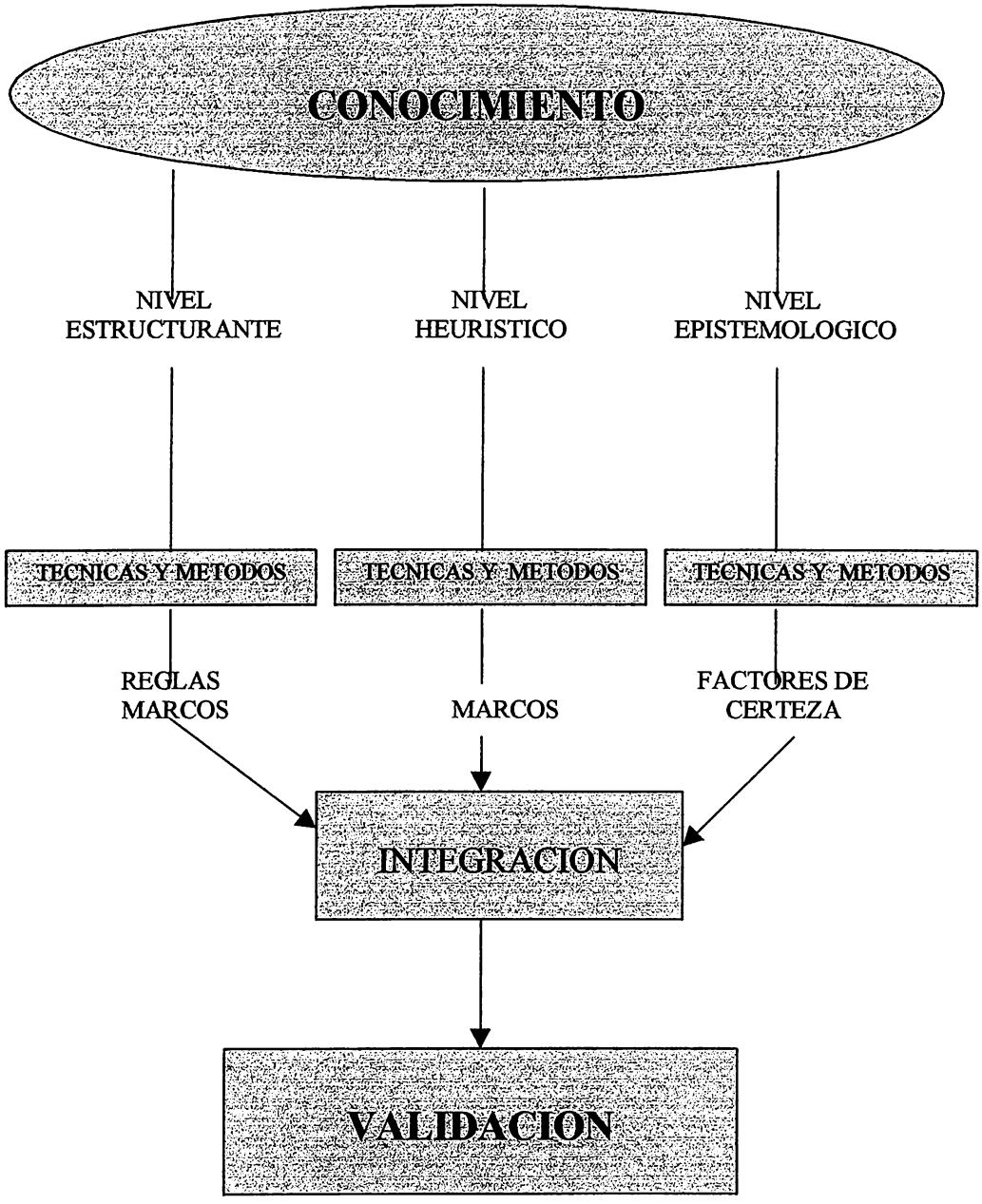
En el nivel estructurante se utilizan las reglas de producción y los frames o marcos.

Las reglas de producción se utilizan en el **nivel heurístico.**

En el **nivel epistemológico** son utilizadas las redes semánticas, los factores de certeza y otros.

Al igual que en el nivel heurístico, en el **nivel conceptual** son utilizadas las reglas de producción.

METODOS Y TECNICAS DE ADQUISICION DEL CONOCIMIENTO



CAPITULO V
INVESTIGACION DE LOS LENGUAJES DE PROGRAMACION

Investigación de los lenguajes de programación referentes a sistemas expertos.

En este capítulo se presentan los aspectos más importantes que debe cumplir un lenguaje de programación de Inteligencia Artificial. Inicialmente se realiza una breve explicación acerca del entendimiento de un lenguaje y los diferentes niveles de abstracción que posee, luego se mencionan los requerimientos específicos que debe tener un lenguaje para que pueda ser utilizado en la programación de Inteligencia Artificial. Finalmente se presentan las generalidades de algunos lenguajes de programación para la Inteligencia Artificial tales como: Visual C++, Visual Basic, Visual Fox, PROLOG, LISP, OPS5, y otros.

5.1 Lenguajes de Inteligencia Artificial.

La función primaria de un lenguaje de Inteligencia Artificial es la de implementar la representación y las estructuras de control necesarias para la computación simbólica. Un lenguaje de Inteligencia Artificial es el encargado de la representación del conocimiento, y no solamente debe expresar el conocimiento requerido para una aplicación, sino que también debe ser conciso, modificable y computacionalmente eficiente y debe asistir al programador en la adquisición y organización de la base del conocimiento.

Teóricamente, cualquier programa de computadora puede ser escrito en cualquier lenguaje de programación. La programación de Inteligencia Artificial típicamente requiere capacidades para el procesamiento simbólico, procesos de inferencia asociadas con los lenguajes, tipo de datos interactivos y sobretodo, flexibilidad.

Los lenguajes para la construcción de programas de Inteligencia Artificial tienen la capacidad para que tome precaución del procesamiento computacional de bajo nivel, de esta manera permite al desarrollador enfocarse en el requisito de complejidad. Los programas de Inteligencia Artificial trabajan con conceptos expresados en palabras, frases o sentencias.

Por lo tanto la habilidad del manejo de datos simbólicos (es decir, no numérico) es una característica importante para desarrollar un sistema de Inteligencia Artificial. Un programador intenta de numerosas maneras implementar cada función componente. En vez de ir a través de un ciclo largo de edición-compilación-depurador para examinar cada versión de una función, un lenguaje de Inteligencia Artificial debería permitir al programador ver rápidamente los resultados de una nueva idea, esta característica de programación de direccionamiento interactivo son necesitados en un lenguaje. El programador no puede anticipar el tipo exacto o la cantidad de datos que fluye a través de un programa, entonces la mayoría se adapta en el camino, por lo tanto un lenguaje debe

incorporar estructuras flexibles de datos que permite que esto pase de manera fácil y natural. Muchos tipos de procesos de inferencia, recurren todo el tiempo a aplicaciones de Inteligencia Artificial, es así como un lenguaje que tiene uno o más de estos procesos construidos internamente pueden ahorrar tiempo y esfuerzos.

Muchos de los algoritmos y estructuras de datos utilizados para implementar lenguajes de Inteligencia Artificial son comúnmente técnicas de la computación tal como arboles binarios y tablas.

Todo lenguaje de programación posee ciertos niveles que son llamados niveles de abstracción, los cuales son herramientas esenciales para entender el comportamiento y la organización de sistemas complejos. Estos niveles pueden ser representados en forma jerárquica tal y como se muestran en la figura 1, donde se observa que el nivel de conocimiento (el cual es el que le compete primordialmente a los lenguajes de Inteligencia Artificial) están en el nivel superior y por lo tanto engloban al resto de niveles. De este esquema podemos señalar otra característica importante, que es la consulta a la base del conocimiento. Los usuarios generalmente ignoran la representación utilizada en el nivel de símbolos y están interesados solamente en el conocimiento del sistema y en la habilidad para resolver sus problemas, es decir que la forma en que el lenguaje opera internamente es totalmente transparente al usuario. De forma similar, en la construcción de la base del conocimiento, el programador es sabedor de las representaciones utilizadas en el nivel de símbolos, pero usualmente ignoran los niveles bajos del sistema. La importancia de estas distinciones es que permiten al usuario o al programador ignorar la complejidad escondida en los niveles bajos.

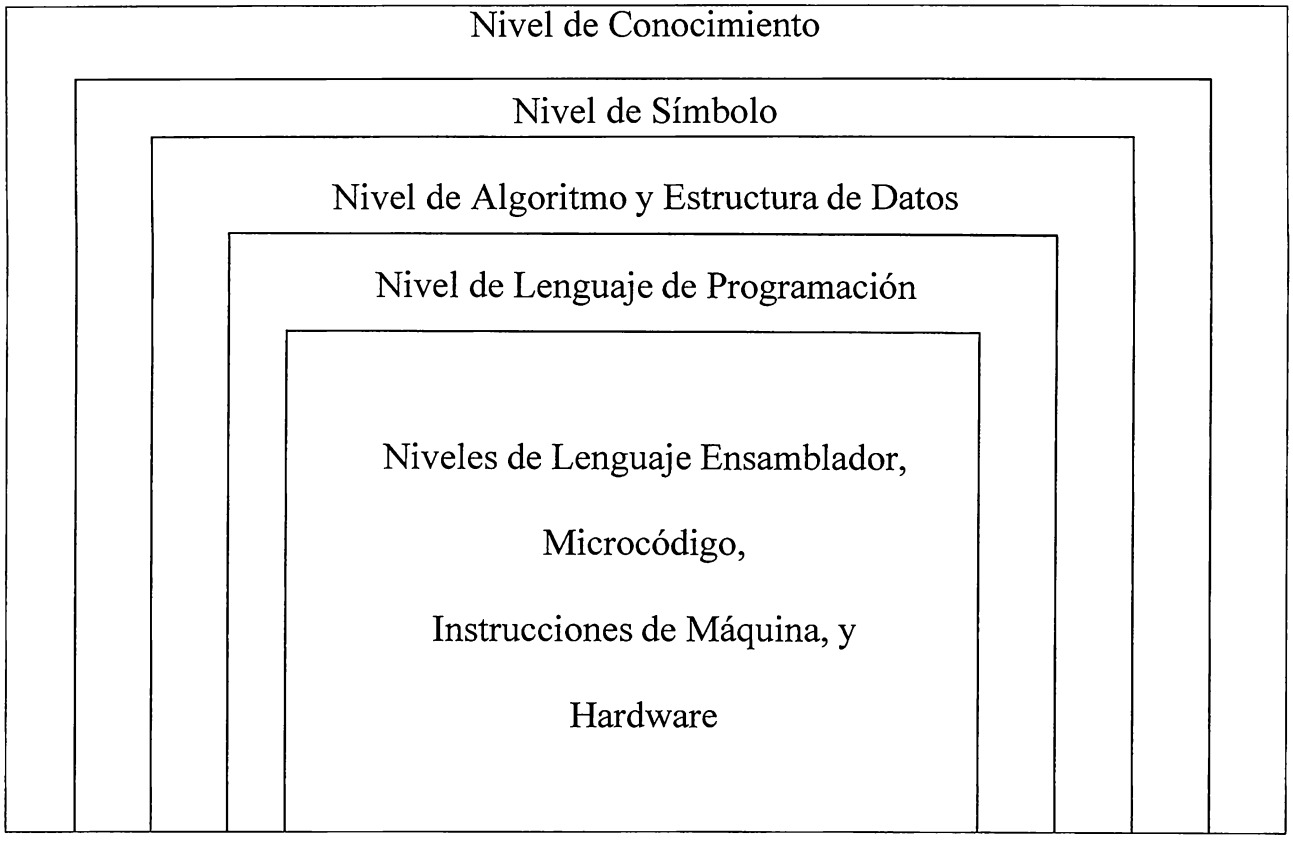


FIGURA 1: niveles de abstracción

Para seleccionar un lenguaje de programación con las características expuestas anteriormente básicamente debe poseer los siguientes requerimientos:

- Soporte de computación simbólica.
- Flexibilidad de control.
- Soporte de metodología de programación exploratoria.
- Asignación tardía y propagación de limitaciones.
- Una clara y bien definida semántica.

- **Soporte de computación simbólica.**

Los tipos y operaciones de datos numéricos enfatizados por los lenguajes de programación tradicionales, no se adaptan bien para la implementación de algoritmos de búsqueda o para la

implementación de lenguajes que representen la Inteligencia Artificial. En lugar de esto, un lenguaje de programación de Inteligencia Artificial debe simplificar la creación de estructuras de símbolos de cualquier tipo y las operaciones en estas estructuras. Este es el requerimiento fundamental para un lenguaje de programación de Inteligencia Artificial.

Un lenguaje de Inteligencia Artificial hace uso de dos poderosas herramientas para la creación de estructuras de símbolos. Una de ellas es el cálculo del predicado, el cual se basa en el álgebra booleana; la lógica booleana es una herramienta poderosa para la construcción de descripciones cualitativas de dominios. La otra herramienta son las listas, las cuales son una secuencia de elementos en la cual cada elemento puede ser incluso otra lista o un símbolo atómico (único). El poder de las listas es en gran parte el resultado de su habilidad para representar cualquier estructura simbólica, sin hacer uso de su complejidad, dimensionalidad o de las operaciones que soporta.

- **Flexibilidad de control.**

Otra distinción del comportamiento inteligente es su flexibilidad. Realmente, críticos de la Inteligencia Artificial han argumentado que la inteligencia no puede ser llevada a cabo por la ejecución paso a paso de secuencias de instrucciones fijas exhibidas por los programas tradicionales de computadoras. Afortunadamente esta no es la única forma para la computadora.

Uno de los más viejos y más importantes paradigmas para la construcción de programas de Inteligencia Artificial es el sistema de producción. En un sistema de producción, el programa es un conjunto de reglas que son ejecutadas en un orden determinado por el modelo de datos en una instancia de un problema dado. Las reglas de producción pueden respaldar virtualmente en cualquier orden a una situación dada. De esta manera, un sistema de producción puede proveer la flexibilidad y la coherencia requerida para un comportamiento inteligente.

Mientras la Inteligencia Artificial utiliza un número diferente de estructuras de control, muchas de ellas están relacionadas con sistemas de producción, y virtualmente todas ellas involucran modelos de pareamientos. Los modelos de dirección de control, permiten que el conocimiento sea aplicado oportunamente en respuesta a las características de una instancia particular de un problema. Los modelos de pareamiento determinan cuando las características de la instancia de un problema, son similares a un trozo de programa de conocimiento, seleccionando ese conocimiento para aplicarlo al problema. Es importante que un lenguaje de Inteligencia Artificial provea esto directamente o simplifique el desarrollo de los modelos de dirección de control.

- **Soporte de metodología de programación exploratoria.**

Los problemas a los que se direcciona la Inteligencia Artificial no siempre responden a algunos planteamientos estándar del software de ingeniería, tal como el diseño top-down, el refinamiento sucesivo y el desarrollo de programas desde especificaciones detalladas formales. Debido a la naturaleza variada de los problemas de Inteligencia Artificial, rara vez es posible dar una correcta y completa especificación de la forma final de un programa de Inteligencia Artificial antes de construir al menos un prototipo. Frecuentemente nuestro entendimiento del problema, el cual el programa trata de solucionar, cambia a través del curso del desarrollo del programa. Las razones para que esto ocurra se mencionan a continuación:

- La mayoría de los problemas de Inteligencia Artificial son en un inicio pobremente especificados.
- Las aproximaciones tomadas para solucionar problemas tienden a ser de un dominio especifico.
- Los métodos heurísticos son inherentemente empíricos.
- La programación basada en el conocimiento parece ser fundamentalmente incremental por naturaleza.

Por estas razones, la programación de Inteligencia Artificial es inherentemente exploratoria. El programa es frecuentemente el vehículo a través del cual nosotros exploramos el problema y descubrimos las estrategias de solución. El reto de la programación de Inteligencia Artificial es el de encontrar formas de realizar programación exploratoria que sea eficiente, sistemática y bien estructurada. Entre las características que un lenguaje de programación exploratoria debe tener, se encuentran las siguientes:

- **Modularidad y modificalidad de código.**

  Es importante que un lenguaje para la programación exploratoria soporte modificaciones frecuentes de su código. Esto implica que el problema debe consistir de pequeños y bien limitados trozos de código más que de grandes cuerpos de código complejo. Esta interacción entre los componentes de un programa deben estar limitados y claramente definidos. Esto incluye revisar efectos laterales y variables globales y asegurarse que el rol de cualquier trozo en la ejecución del programa es fácilmente determinado.

- **Extensibilidad y soporte de lenguajes.**

  La programación exploratoria procede del bottom-up, con estructuras de alto nivel emergiendo al mismo tiempo del desarrollo del código de un programa. Frecuentemente, mientras no es posible especificar la forma final de un programa de Inteligencia Artificial, es posible determinar las estructuras de alto-nivel útiles para la exploración del ámbito del problema. Estas estructuras pueden incluir modelos de pareamiento, controles de búsqueda y funciones para definir un lenguaje de representación. Esencialmente este enfoque dice "si no se puede determinar la estructura final que un programa va a tener, intente definir las construcciones de lenguaje que ayudan a desarrollar esa estructura".

Para soportar esa metodología, un lenguaje de programación debe ser fácilmente extensible y debe simplificar el desarrollo de interpretes. Por extensibilidad, nos referimos a la habilidad de definir nuevas construcciones de lenguaje con la máxima libertad y flexibilidad. Una vez definido, estas construcciones creadas por el usuario se comportan exactamente igual que los componentes construidos internamente del lenguaje. El ingeniero del conocimiento programa estos lenguajes extendiendo sus capacidades básicas desde abajo hacia arriba, hasta que la solución sea alcanzada. En este sentido los programas tradicionales son "construidos", pero los programas de Inteligencia Artificial van "creciendo".

- **Existencia de construcciones útiles de alto nivel.**

  La programación es asistida por la existencia de poderosas construcciones de alto nivel en el lenguaje. Estas construcciones poderosas pero en general abstractas permiten al programador desarrollar rápidamente estructuras específicas y especializadas para la representación del conocimiento y control de programa.

- **Soporte de prototipo de funciones y objetos.**

  Otra metodología importante de la programación exploratoria es el prototipo de funciones y objetos. Aquí, el programador construye una solución al problema y la utiliza para explorar el espacio del problema. Una vez se ha explorado el problema y delimitado una estrategia de solución viable, se desecha el prototipo y se construye un programa final que enfatiza la eficiencia y certeza de la implementación. Las estructuras y los métodos que proveen los lenguajes de Inteligencia Artificial aumentan la velocidad de desarrollo de prototipos.

- **Legibilidad y documentación de programas.**

  Debido a que la mayoría de los programas de Inteligencia Artificial son modificados extensiblemente a través de su tiempo de vida, es importante que el código sea legible y bien

documentado. Mientras que no haya un substituto para los comentarios claros en lenguaje natural, los lenguajes de Inteligencia Artificial por tener código altamente modular compuestos por estructuras de alto nivel, simplifican grandemente esta meta.

- **Interpretación versus compilación.**

  En un principio la mayoría de los lenguajes de Inteligencia Artificial eran interpretes en lugar de compiladores durante el desarrollo del programa. Esto significa que el programador no tenía que esperar largas recompilaciones cada vez que el código era cambiado. Pero desde que la interpretación de código se ejecutan lentamente, los lenguajes de Inteligencia Artificial modernos incluyen facilidades de depuración y rastreo; además siguen haciendo uso de la interpretación de código y permiten la compilación de versiones finales de programas.

- **Software soportado por la programación exploratoria.**

  Los lenguajes de Inteligencia Artificial modernos incluyen diversos ambientes de programación, proveyendo herramientas para rastrear la ejecución ya sea de todos los programas o secciones de programas. Los depuradores permiten al programador avanzar a través de la ejecución del programa incrementalmente, para cambiar temporalmente el valor de las variables de programa o incluso el código del mismo, para insertar puntos críticos que obligan a detener la ejecución de programas en puntos pre-específicos y congelar el ambiente de ejecución en el punto donde un error es detectado y así examinar el estado del mismo. En adición, muchas implementaciones de lenguajes incluyen editores inteligentes que muestran los errores de sintaxis como si el código estuviese siendo escrito. Debido a la complejidad de los programas de Inteligencia Artificial y la dificultad de predecir el comportamiento de la mayoría de los regímenes flexibles de control como los sistemas de producción, lo importante de estas características no se debe pasar por alto.

- **Asignación tardía y propagación de limitaciones.**

La propagación de limitaciones es una metodología importante de programación de Inteligencia Artificial que debe ser soportado por un lenguaje de implementación. Frecuentemente los problemas direccionados por programas de Inteligencia Artificial requieren que los valores de ciertas entidades permanezcan desconocidos hasta que la suficiente información ha sido reunida para determinar asignamiento. Esta información tiene que ser vista como una serie de limitaciones en los valores que una variable pueda asumir. Así como las limitaciones se van acumulando, el conjunto de posibles valores se va reduciendo, hasta que se convierta en soluciones que satisfagan todas las limitaciones. Un simple ejemplo de este enfoque, puede ser visto en un Sistema de Diagnóstico Médico que reúne información acerca de los síntomas del paciente hasta que las posibles explicaciones van siendo reducidas a un solo diagnóstico. La analogía de esta metodología a los lenguajes de programación es la variable explícitamente ilimitada mientras es manipulada en el código del programa. Esto permite la fácil y natural implemetación de propagación de limitantes.

- **Una clara y bien definida semántica.**

Los lenguajes de Inteligencia Artificial comparten con los lenguajes de programación orientados al desarrollo de largos compiladores y un sistema realizable, la necesidad de una clara y bien definida semántica. Desafortunadamente, lenguajes de programación como Fortran y Pascal tienden a tener definiciones de semántica complejas y difíciles. Este defecto puede ser relacionado con el hecho de que estos lenguajes son esencialmente caracterizaciones de alto nivel de la arquitectura fundamental de la computadora de Van Newman y heredan muchas complejidades de ese sistema físico. Desde que los lenguajes de Inteligencia Artificial han sido basados en formalismos matemáticos como la lógica o en la teoría de funciones recursivas, estos tienden a tener semántica simples, heredando bastante el poder rotacional de elegancia de la matemática formal. Esto hace que estos lenguajes

sean particularmente útiles para búsqueda en áreas como implementación de lenguajes de representación del conocimiento, formalizando el proceso de desarrollo de código, proveyendo programas correctos y automatizando la generación de código eficiente desde especificaciones formales. Estas áreas de búsqueda requieren que la semántica de los lenguajes de programación sean claramente y formalmente especificadas.

También debe ser notado que aunque la función de la mayoría de los programas de Inteligencia Artificial es altamente compleja, el código que implemente esa función debe esforzarse por la simplicidad y claridad.

5.2 Investigación de los lenguajes de programación para sistemas expertos

Antes de que existieran sistemas expertos, sistemas inteligentes adaptables o cualquier otro tipo de programa capaz de funcionar con inteligencia artificial, se necesitó otro tipo de programa capaz de funcionar con inteligencia artificial, se necesitó crear los lenguajes para desarrollarlos. Para ello, se consideraron algunos requerimientos básicos como la posibilidad de procesar símbolos de todo tipo y la capacidad de hacer inferencias asociadas con el lenguaje, todo dentro de un ambiente flexible que permitiera escribir el programa de forma interactiva.

Para crear programas inteligentes, existen lenguajes "viejos" como el EIFFEL, LISP, LOGO, OPS, PROLOG; los no tan "viejos" como DYLAN, FORTH, VISUAL BASIC,VIASUAL C, VISUAL FOX, entre muchos otros más. Cada uno trabaja de forma distinta: algunos utilizan vectores, objetos (aglomeraciones de software) o lógica, lo que los hace mejor capacitados para diversas tareas.

PROLOG

- **Soporte de computación simbólica:**

  Los programas en PROLOG son colecciones de hechos descriptivos acerca de un dominio y reglas para la derivación de conclusiones desde aquellos hechos. Ya que PROLOG esta basado en una concisa notación matemática, los programas tienden a ser compactos, capturando el conocimiento para la solución de problemas en unas pocas oraciones.

  Por ser una implementación de la lógica formal, PROLOG es algunas veces utilizado directamente como un lenguaje de representación en el nivel de simbolos. De cualquier forma, su verdadero poder es como un lenguaje para la implementación de representaciones más complejas y especializadas tales como estructuras (frames) y redes en una forma concisa y sistemática. Muchas estructuras del nivel de simbolos son facilmente construidos utilizando construcciones de alto nivel de PROLOG.

- **Flexibilidad de control:**

  En PROLOG la unificación y los algoritmos de busqueda están construidos dentro del lenguaje mismo, es simple el construir cualquier régimen de control de modelos de direccionamiento: la primera busqueda profunda (depth-first) dada por omisión y otros regímenes tales como la primera busqueda a lo ancho (breadth-first search) o la primera-mejor busqueda (best-first search) deben ser construidas con pocas lineas de codigo.

- **Soporte de metodologías de programación exploratorio:**

- Modularidad y modificalidad del código:

  En PROLOG la unidad básica de un programa es la regla; estas reglas, al igual que las funciones de LISP, tienden a ser pequeñas y especializadas. Desde que el alcanze de las variables en PROLOG es siempre restringido por una simple regla y el lenguaje no permite variables

globales, la modificalidad es simplificada. PROLOG ofrece facilidades de rastreo, los cuales, cuando se combinan con una clara estructura de programa, simplifica el asignamiento de las culpas y los créditos en la depuración.

- Extensibilidad y soporte de lenguajes:

  PROLOG simplifica la escritura de interpretes. Provee esta capacidad a través de los llamados "meta-predicados" (meta-predicates), los cuales son predicados para la manipulación de otros predicados de PROLOG, nuevamente simplificando la escritura de interpretes arbitrarios. Al igual que con LISP, un gran numero de lenguajes para Inteligencia Artificial de alto nivel han sido construidos sobre PROLOG utilizando esta metodología.

- Existencia de construcciones útiles de alto nivel:

  PROLOG ha seguido siendo un pequeño lenguaje comparativamente, en parte por ser novedoso y por su compromiso de ser compacto y simple. De cualquier modo, PROLOG permite a sus usuarios crear sus propias librerías de predicados especializados, y las más útiles de estas librerías han encontrado su camino dentro de las implementaciones estandar.

- **Asignación tardía y propagación de limitaciones:**

  PROLOG permite que variables sean definidas y manipuladas con ilimitaciones explicitas, mientras se están definiendo las relaciones y las dependencias entre esas variables y otras unidades de programa. Esto permite la fácil y natural implementación de la propación de limitaciones.

- **Generalidades:**

  PROLOG es considerado como la puerta de los lenguajes de programación ya que soporta razonamiento simbólico formal, haciendo teóricamente posible "computadoras inteligentes" que pueden entender el lenguaje humano y diagnosticar enfermedades médicas.

El primer programa fue escrito en Marsella Francia, a principios de 1970 como parte del entendimiento natural. El fondo teórico del lenguaje PROLOG tiene raíces antiguas, especialmente en el trabajo de Kowalsky, Hayes y otros.

La primera versión de PROLOG fue desarrollada por los años 1975-1979 en el departamento de Inteligencia Artificial de la Universidad de Edinburgh por David H. D. Warren y Fernando Pereira. Ellos introdujeron el primer interprete robusto de PROLOG.

PROLOG puede se ejecutado en los siguiente entornos: Sistema Operativo DOS, Windows 3.1, Windows 95, Windows NT. Además actualmente se estan desarrollando versiones para la plataforma Machintoch.

LISP

- **Soporte de computación simbólica:**

  Una herramienta importante para la construcción de estructuras simbólicas es la lista. Una lista es una secuencia de elementos en la cual cada elemento puede ser, ya sea otra lista o un símbolo atómico.

  El poder de las listas es el resultado de su habilidad para representar cualquier estructura de símbolos, a pesar de su complejidad, dimensionalidad o las operaciones que soporte.

- **Flexibilidad de control:**

  LISP no provee directamente un algoritmo de modelo de pareamiento (matching), pero lo sofisticado de sus facilidades de computación simbólica y la fácil extensibilidad, del lenguaje lo hacen simple para escribir modelos de pareamiento e interpretes de arbitraria complejidad y organización. Una ventaja de esta aproximación es que el modelo de pareamiento y las

estructuras de control asociadas pueden ser fácilmente hechas a la medida para responder a las demandas de un problema en particular o una representación.

- **Soporte de metodologías de programación exploratoria:**
- Modularidad y modificalidad de código

  Los programas de LISP son escritos como colecciones de funciones individuales; en un programa de LISP bien escrito, cada función es pequeña y ejecuta una sola y bien definida tarea. Así, es usualmente simple el localizar y corregir la causa de cualquier deficiencia. El parametro que es pasado y las reglas de alcance de las variables de LISP, incluso sirven para reducir los efectos laterales de las funciones. Las variables globales, mientras son soportadas por el lenguaje, son evitadas en un buen código de LISP.
- Extensibilidad y soporte de lenguajes:

  LISP simplifica la escritura de interpretes, en él, tanto los programas como los datos son sintácticamente representados como listas, esto lo hace muy fácil para escribir programas que manipulen el código de LISP como si fueran datos, simplificando grandemente el desarrollo de interpretes. Muchos lenguajes de Inteligencia Artificial que son importantes, tanto históricamente como comercialmente, tales como Planner, Rosie, Kec y Ops fueron escritos sobre estas capacidades de LISP.
- Existencia de construcciones útiles de alto nivel:

  En LISP, esto incluye el tipo básico de datos, la lista, la cual permite la construcción de estructura de datos arbitrariamente complejas, tan buenas como las funciones poderosas para definir operaciones en estas estructuras. Dado que LISP es extensible y ha sido utilizado por muchas décadas, las más generales y poderosas de estas funciones definidas por el usuario se han convertido en características estandar del lenguaje. La mayoría de LISPs, han evolucionado

dentro de esta moda para incluir, literalmente, cientos de funciones para la creación estructuras de datos, construir interfaces de usuario, rastreo de la ejecución de programas y edición de las estructuras de LISP.

- **Asignación tardía y propagación de limitaciones:**

  LISP se comporta de la misma forma que PROLOG en este requerimiento.

- **Generalidades:**

  LISP fue inventado por John McCarthy a finales de 1950. El lenguaje fue pasado originalmente como un modelo de computación alternativo, basado en la teoría de funciones recursivas. En un principio, MaCarthy (1960) especificó sus objetivos: crear un lenguaje simbólico computacional más bien que numérico computacional; para implementar un modelo de computación basado en la teoría de funciones recursivas, para proveer una clara definición de la sintaxis y semántica del lenguaje, y para demostrar formalmente la perfección de este modelo computacional. LISP es un acrónimo para "LISt Processing". LISP es uno de los más viejos lenguajes que aun existen, originalmente fue un lenguaje bastante simple y pequeño, consistiendo de funciones para la construcción y acceso a listas, definiendo nuevas funciones.

  LISP puede ser ejecutado en varios entornos tales como: Sistema Operativo DOS, Windows 3.1, Windows 95.

LENGUAJES ORIENTADOS A OBJETOS:

- **Soporte de computación simbólica:**

  Los programas escritos en lenguajes orientados a objetos, cumplen con este requerimiento fundamental para la programación de Inteligencia Artificial, al hacer uso de listas y arboles binarios para la representación de estructuras simbólicas; así como de técnicas y métodos de la

teoría de programación orientada a objetos. No todas las estructuras simbólicas pueden ser representadas a través de estos elementos que proporciona el lenguaje, pero el programador puede hacer uso de las ventajas que los objetos poseen para construir nuevos métodos que sean capaces de realizar las operaciones internas con las que deben ser tratadas las estructuras simbólicas y también puede construir nuevas estructuras independientes y cuyos comportamientos esta regido por los métodos, así como los lenguajes tradicionalmente utilizados para Inteligencia Artificial ofrecen estructuras y procedimientos prefabricados que las manipulan.

- **Flexibilidad de control:**

  Los lenguajes orientados a objetos, ofrecen flexibilidad de control al momento de manipular las estructuras que permiten modelar la representación de conocimiento, entre ellos, algoritmos de búsqueda, algoritmos de indexación. En ningún momento las herramientas de estos lenguajes ocupan una secuencia fija de pasos a seguir, sino que estos pasos los lleva a cabo mediante la invocación de métodos a través del paso de mensajes entre objetos.

- **Soporte de metodología de programación exploratoria**
- Modularidad y modificalidad

  El código en estos lenguajes frecuentemente puede ser modificado, ya que este es escrito en pequeños trozos y no como un gran cuerpo complejo. La facilidad que presentan estos lenguajes al permitir la modularidad, es que se puede tener un control mas completo de lo que cada trozo de código va a realizar y así, es usualmente más fácil el localizar y corregir deficiencias.

- Extensibilidad y soporte de lenguajes

Los lenguajes orientados a objetos, por su naturaleza son extensibles ya que poseen la habilidad de definir nuevas construcciones de lenguaje, es decir, que permiten definir fácilmente nuevos objetos.

- Existencia de construcciones útiles de alto nivel

  La programación a través de lenguajes orientados a objetos permite construcciones poderosas de alto nivel al hacer uso de los objetos. El usuario puede desarrollar rápidamente estructuras especificas que se comportarán de la misma forma con las ya existentes.

- **Una clara y bien definida semántica**

  En los lenguajes orientados a objetos existe una clara y bien definida semántica, es decir, que utilizan de una forma coherente las estructuras proporcionadas por el lenguaje.

VISUAL C++

- **Generalidades:**

  Visual C++ es un programa desarrollado por Microsoft. Ofrece una nueva forma de crear aplicaciones de Windows ya que combina la potencia de la programación orientada a objetos. Visual C++ crea aplicaciones del mundo real para Windows, utilizando una amplia gama de ejemplos de código fuente. Este lenguaje de programación puede ser ejecutado en cualquiera de los entornos que se detallan a continuación:

  - Windows 3.1.
  - Windows95.

  El compilador de Visual C++ puede procesar tanto código fuente en C como también código fuente en C++. El compilador tiene capacidad de determinar cual es el lenguaje en cuestión examinando la extensión de nombre de archivo del código fuente.

El enlazador de Visual C++ procesa archivos con extensión OBJ que produce el compilador. El compilador de Visual C++ opera ya sea en modo compilador o en modo ligar. En modo Ligar, el archivo con extensión RES se fusiona con un archivo ejecutable (es decir, con extensión EXE). Si después es utilizado un archivo RES se puede volver a ligar al archivo EXE sin tener que volverlo a enlazar.

VISUAL BASIC

- **Generalidades:**

  Visual Basic es una de las mejores y sencillas herramientas para crear una aplicación en Windows, independiente de la experiencia que posea el usuario en programación o en el manejo de Windows. El proceso se realiza con una velocidad notable.

  Visual Basic es una herramienta para el desarrollo de software con aplicaciones de bases de datos, creada por Microsoft, con un soporte en la programación orientada a objetos.

  En la creación de un programa en Visual Basic, el primer paso fundamental consiste en plantearse la presentación del programa y el diálogo entre hombre y máquina. El trabajo con Visual Basic no se inicia con un editor de programas, sino con el diseño de interfaz del programa. Luego de establecer los elementos fundamentales de entrada y salida se procede a la programación paso por paso de las funciones.

  Visual Basic esta diseñado para direccionar aquellos requerimientos únicos para la integración de todas las herramientas de soporte necesitados, para el desarrollo, depuración, integración y soporte de sistemas.

  Visual Basic es ejecutado en cualquiera de los siguientes entornos:

  - Windows 3.1

- Windows para trabajo de grupo (si se va a usar la versión 16 bits de Visual Basic y se esta limitado al desarrollo de programas de 16 bits).
- Windows 95 (para crear aplicaciones de 16 y 32 bits).
- Windows NT 3.51 (con al menos 8 megabytes de memoria).

Cuando se ejecutan aplicaciones en Visual Basic no se compila su código. En otras palabras no traduce a un lenguaje de máquina, en vez de eso simboliza o lo traduce a un formato interno más compacto y fácil de manipular.

VISUAL FOX

- **Generalidades:**

Visual Foxpro es una herramienta para el desarrollo de software con aplicaciones de bases de datos, creada por Microsoft, con soporte en la programación orientada a objetos, que es la principal característica que lo diferencia de versiones anteriores. Algunas áreas de Visual Foxpro han quedado prácticamente intactas de su predecesor Foxpro 2.6 para Windows. El diseño de pantallas o formularios en Visual Foxpro es radicalmente diferente.

Da a los desarrolladores las herramientas necesarias para el manejo de datos, organizar tablas de información y correr consultas creando un sistema de base de datos relacionales (DBMS) o la programación de una aplicación de manejo de datos para usuarios finales.

Visual Foxpro es un entorno de desarrollo para escribir aplicaciones de bases de datos. Proviene de la generación Xbase, de lenguajes de programación que incluyen dBase III, Foxbase, Foxpro.

Visual Foxpro es un lenguaje de programación con soporte en la programación a objeto, la cual es un mejor sistema para el desarrollo de aplicaciones. Una de las mejoras que se obtienen con el uso de la programación orientada a objetos es el permitir afrontar programas más complejos y de

mayor tamaño con menos esfuerzos. Además, mejora considerablemente el rendimiento de desarrollo. También otorga una gran flexibilidad y es más sencillo para realizar mantenimiento en los programas.

Existen varios ambientes dentro de los cuales puede ejecutarse Visual Foxpro, entre los cuales se pueden mencionar:

- Windows 3.x.
- Windows 95.
- Windows NT.

Cuando se ejecutan aplicaciones en Visual Foxpro no se compila su código, en otras palabras no traduce a un lenguaje de máquina; en vez de eso simboliza o lo traduce a un formato interno más compacto y fácil de manipular.

DYLAN

- **Generalidades:**

  DYLAN es un nuevo lenguaje dinámico orientado a objetos (OOAL), que fue desarrollado por APPLE. Este lenguaje tiene la meta de desarrollar una herramienta práctica para la escritura de las principales aplicaciones comerciales. Su propósito es la de combinar las mejores cualidades de los lenguajes estáticos (pequeños programas rápidos) con las mejores cualidades de los lenguajes dinámicos (rápido desarrollo, código que es fácil de leer, escribir y mantener). Difiere de Visual C++ en muchas maneras importantes que lo hacen poderoso y flexible. DYLAN incluye numerosas características que lo distinguen de Visual C++ tales como:

  - Manejo automático de memoria.
  - Sintaxis clara y consistente.

- Modelo total y consistentemente orientado a objetos.
- Soporte de compilación incremental.
- Funciones y clases de primera clase.

Los programas de DYLAN son modulares. Este sistema modular soporta múltiples interfaces con control preciso sobre importaciones y exportaciones, así los clientes están garantizados que verán solamente aquellos objetos que son apropiados.

El objetivo actual de DYLAN son las plataformas de Windows 95 y Windows NT, para los cuales provee compilación nativa, soporte de OLE y COM. Implementaciones para otras plataformas serán anunciados dentro de poco tiempo.

DYLAN no provee una nueva plataforma o soporta el menor denominador común a través de las plataformas. En lugar de esto, DYLAN provee herramientas para el total acceso a la plataforma en la cual es el anfitrión. DYLAN tiene la meta de crear código bastante portable, a través del uso de abstracciones de alto nivel. DYLAN ayuda también al WEB como un lenguaje poderoso para implementar utilitarios de Server y como un lenguaje de alto nivel para implementar partes de Active X.

DYLAN fue diseñado para que fuera de propósito general. Lenguaje totalmente orientado a objetos para utilizar en sistemas, aplicaciones y componentes de programación. El lenguaje fue diseñado para ser simple y poderoso, internamente consistente y de grandes características.

Todos los datos en los programas que están corriendo son objetos, incluyendo clases, funciones, métodos y números. Todos los objetos son instancias de clases, todos pueden ser pasados a métodos, almacenados en variables, etc.

DYLAN utiliza tipos de inferencia para compilar objetos en mas tipos primitivos, por medio de eso logra la eficiencia sin alterar el modelo objeto.

DYLAN es un lenguaje seguro, no hay apuntadores explícitos, las operaciones son chequeadas antes de ser ejecutadas (en tiempo de compilación o en tiempo de corrida). En general DYLAN garantiza que el modelo objeto no es violado por algunas cosas como referencia a arreglos prohibidos.

DYLAN utiliza una sintaxis basada en expresiones fáciles de leer que enfatizan el uso de palabras, como el que se necesita para crear código claro. La mayoría de los programadores tienen pocos problemas en entender el código de DYLAN, incluso cuando no están familiarizados con el lenguaje.

DYLAN toma ventaja de la garantía de sus directivas para compilar estáticamente código orientado a objetos. Esto da como resultado un código nativo eficiente, sin necesidad de sacrificar el diseño pero orientado a objetos. Este compilador optimiza tanto el espacio como la velocidad.

EIFFEL

- **Generalidades:**

EIFFEL es un lenguaje avanzado orientado a objetos creado en 1985 por Bertrand Meyer. Nombrado en honor a Gustavo EIFFEL, el ingeniero de diseño que diseño la torre EIFFEL, EIFFEL fue desarrollado por interctive Software Engineering (ISE) de Gotelg, California. Es un avanzado lenguaje de programación orientado a objetos, que enfatiza el diseño y el desarrollo de la calidad y la realización del software. A diferencia de Visual C++, este lenguaje no esta basado en la extensión de cualquier otro lenguaje. EIFFEL restringe prácticos peligros de generaciones previas de lenguaje, incluso establece una interface con otros lenguajes tales como C, C++,

Visual C. EIFFEL acoge el concepto del "Diseño por Centrado" para la realización correcta del software. EIFFEL esta basado en varios conceptos:

- Herencia múltiple, incluyendo redefinición, indefinición, recomendación, selección.
- Asesorías para escribir, depurar y documentación del software automáticamente
- Desarrollo parecido al que entra en el ciclo de vida entero, desde el análisis y diseño de alto nivel hasta la implementarción y el mantenimiento.
- Clases, sirviendo tanto para la estructura de módulo como para el tipo de sistema
- Herencia para la clasificación, tipo, subtipo y rehuso de software
- Manejo de excepciones para recobrarse de los estados no planeados

EIFFEL es totalmente portable, es adaptable sobre muchas plataformas con completa fuente de compatibilidad. Es decir que la misma semántica puede ser soportada en diferentes plataformas.

EIFFEL puede ser utilizado en diferentes áreas, ha sido probado exitosamente en grandes proyectos orientados a objetos, tales como: Finanzas, Telecomunicaciones, CAD-CAM, CASE, educación, etc.

Utiliza técnicas de compilación usando C como el lenguaje intermediario y generador de ejecutables cuyas velocidades se equiparen o excedan a C o FORTRAN.

FORTH

- **Generalidades:**

Fue creado por Charles Moore entre 1960 y 1970 para dar a los computadores control en tiempo real sobre el equipo astronómico. Un numero variado de características de FORTH (incluyendo su estilo interactivo) lo convierte un lenguaje útil para la programación de Inteligencia Artificial, y algunos devotos han desarrollado sistemas expertos y redes neurales basadas en FORTH.

Dos grupos gubernamentales (The Form Interest Group y The Institute for Applied FORTH Research) ayudaron a promocionar el lenguaje. Dos libros escritos por Brodie (1984, 1987) son quizás las mejores presentaciones de FORTH que se conocen, y un articulo de Sperry (1991), es un vistazo corto y bien informado.

Las funciones en FORTH son llamadas "palabras". El programador utiliza las palabras empotradas de FORTH para crear otras nuevas y almacenarlas en el "diccionario" de FORTH. En un programa de FORTH, las palabras pasan información hacia otra colocando datos en (y removiendo datos de) una "pila", una estructura de Software en la cual el ultimo elemento en entrar es el primero en salir. Utilizando una pila de esta forma, permite a FORTH correr aplicaciones rápidas y eficientemente.

LOGO

- **Generalidades:**

Ideado a finales de 1960 por Popert y sus colegas en el MIT (Massachussets Institute, Instituto de Massachussets) como una ayuda educacional para niños. LOGO es un subconjunto de LISP y puede ser usado como un serio lenguaje de programación. LOGO es compacto, y esto lo hace un excelente vehículo para le exploración de la Inteligencia Artificial en una PC.

Una versión de LOGO existe virtualmente para la mayoría de las marcas de las computadoras personales con versiones en la mayoría de los casos etiquetados por el fabricante de la computadora.

LOGO es mejor conocido por sus gráficos; el usuario programa un cursor (llamado un "Turtle") para dibujar figuras en la pantalla. Dentro de los gráficos, LOGO tiene un gran numero de funciones de manipulación de listas (en una sintaxis que es mas amigable que la de LISP) que lo

hacen ideal para aplicaciones de la Inteligencia Artificial y también posee funciones de generación de números aleatorios que lo hacen controlable para simulaciones.

OPS

- **Generalidades:**

A los científicos de la Universidad de Carnegie-Melloon, Herbert Simon y Allen Newell, investigaron los procesos de resolución de problemas humanos por muchas décadas. Una de sus ideas, el sistema de producción, representa al conocimiento como un conjunto de reglas de condición y acción (también llamadas reglas "If-then"). Si las condiciones para una regla eran sofisticadas, ocurría una acción especificada por la regla. A finales de 1970, Charles Forgy desarrolló OPS5, un lenguaje de programación que incorpora esta idea. El sistema de producción vino a ser la metodología dominante de la representación del conocimiento en Sistemas Expertos, y OPS5 vino a ser el más popular entre los desarrolladores de Sistemas Expertos.

Las reglas en OPS5 son totalmente independientes una de otra, ellas pueden ser colocadas en la memoria de producción en cualquier orden. Si los datos en la memoria de trabajo encajan con las condiciones de una regla en la memoria de producción, la acción de la regla toma lugar. Posibles acciones incluyen la modificación desplegada en la pantalla, y llamadas a programas externos. En algunas versiones del lenguaje, las acciones de las reglas pueden causar que se creen nuevas reglas, permitiendo el desarrollo de sistemas que aprenden.

El mecanismo de inferencia construido internamente de OPS5, el forward chaining (encadenamiento hacia delante) comienza de un conjunto de datos que se incrementan en la memoria de trabajo, causando una secuencia de reglas que están siendo aplicadas hasta que la meta es alcanzada. El otro procedimiento de inferencia de Sistemas Expertos es el backward

chaining (encadenamiento hacia atrás), en el cual una meta es alcanzada causando submetas o se satisface hasta que una solución al problema es encontrada.

CAPITULO VI
METODOLOGIAS DE LA INVESTIGACION

Metodología de la investigación.

El capítulo presenta la tipología de investigación que se ha realizado, la cual en un inicio se realizó llevando a cabo una investigación exploratoria, a continuación se llevo a cabo una investigación de tipo descriptiva. Además se describen las diferentes técnicas utilizadas para la adquisición del conocimiento, así como la metodología de desarrollo que comprende el análisis y el diseño del prototipo. Finalmente se describe la herramienta de programación bajo la cual se ha desarrollado el software.

6.1 Metodología de la investigación

6.1.1. Tipo de investigación.

Para el desarrollo de la investigación se realizaron diferentes metodologías, las cuales se describen a continuación:

6.1.1.1 Estudio Exploratorio.

Un estudio exploratorio fue necesario en la etapa inicial del proyecto, mediante la técnica de la encuesta, para llegar a identificar el grado de conocimiento sobre la inteligencia artificial y los sistemas expertos, tanto de las personas que estudian computación como de los que se han preparado en áreas afines, e identificar en que áreas son más utilizados los Sistemas Expertos. El resultado que arrojo este estudio es que un 80 % de las personas encuestadas no tenían conocimiento de la existencia de la inteligencia artificial y por consiguiente no habían escuchado el termino Sistemas Expertos, el 20 % que tenia conocimientos eran en su mayoría personas graduadas de licenciatura o ingeniería en computación pero con un grado de maestría obtenida en el extranjero donde escucharon hablar sobre la inteligencia artificial y una de sus técnicas, la cual es Sistemas Expertos.

6.1.1.2 Estudio Descriptivo.

Durante la investigación se hizo necesario utilizar un estudio descriptivo o Bibliográfico para detallar aspectos generales de lo que es la Inteligencia Artificial, su definición, el área de investigación, los subcampos en que esta dividida. Este estudio llevó a un análisis exhaustivo de lo que son los Sistemas Expertos, su historia, los campos de aplicación, los componentes de los Sistemas Expertos, el equipo de desarrollo, la forma de representación del conocimiento.

6.1.1.3 Estudio sobre lenguajes de programación.

Luego se realizo una investigación de los lenguajes de programación referentes a Sistemas Expertos, a fin de determinar los requerimientos necesarios para la selección de un lenguaje de programación de Inteligencia Artificial.

La investigación descriptiva se hizo posible con la recopilación de información de diferentes fuentes tales como: consultas a libros de texto, uso de revistas científicas, consultas a los expertos, uso de tesis, consultas a Internet.

6.2 Técnicas e instrumentación de investigación.

Para el desarrollo de Sistemas Experto se han utilizado diferentes técnicas, las cuales se hacen necesarias para la construcción de un sistema basado en el conocimiento. Debido a que se necesita la experiencia de un experto, se hizo necesario el uso de la entrevista, las cuales se han realizado a diferentes expertos a fin de extraer de ellos definiciones, hipótesis, y experiencias que son necesarias para la resolución del problema.

Las entrevistas realizadas a los diferentes expertos son presentadas en el anexo A de este documento.

También otra de las técnicas utilizadas ha sido la búsqueda de literatura especializada para complementar la base de conocimientos, la cual sirve para reforzar la información adquirida de los expertos e incluida en la base del conocimiento.

6.3 Metodología de desarrollo del Sistema Experto.

6.3.1 Etapa de análisis.

Para la etapa del análisis del problema se han evaluado los diferentes requerimientos comunes al dominio del problema, el cual es la construcción de un prototipo de prótesis del miembro inferior. Para esto, se han realizado diferentes actividades que permiten mostrar la secuencia para la elaboración del prototipo de diagnóstico de prótesis del miembro inferior.

6.3.2 Etapa de diseño.

La etapa de diseño presenta gráficamente la secuencia de cómo esta constituido el programa, en esta etapa se representan las decisiones, procesos y flujos de información utilizados para el buen funcionamiento del sistema y poder así obtener los resultados deseados.

6.3.3 Etapa de programación.

Después del diseño para la elaboración del prototipo se pasa a la etapa de programación, inicialmente se uso Turbo Prolog, pero por no cumplir con las expectativas de ambiente necesarias de hoy en día de poseer una interfaz amigable con el usuario, se decidió utilizar Win-Prolog y Shell llamado Flex, el cual es un poderoso y expresivo sistema experto que soporta el razonamiento basado en marcos o frames con herencias, la programación basada en reglas y procedimientos basados en los datos y esta totalmente integrado dentro del ambiente de programación lógica, y contiene su propio lenguaje llamado KSL (English-like Knowledge Specification Language, Lenguaje se especificación del conocimiento como el inglés). Tomando en cuenta los requerimientos básicos que debe tener un lenguaje de programación para Inteligencia Artificial, y aprovechando las ventajas que posee WIN-PROLOG a nivel de Windows ya que trabaja directamente con los 32 bits de la API (Interfaces de Programación de Aplicación) de Windows NT

y Windows95, y provee el acceso conveniente a un gran número de funciones GUI (Interface Gráfica del Usuario de Windows), sé decidió realizar el prototipo con este lenguaje para sacar provecho de la capacidad de creación de una interface amigable para el usuario unido a la capacidad del motor inferencial que posee internamente el lenguaje y así construir un prototipo de aplicación que simule los conocimientos de un experto.

CAPITULO VII
PASOS A SEGUIR PARA CONSTRUIR UN PROGRAMA DE INTELIGENCIA ARTIFICIAL EN CUALQUIER AREA DE APLICACION

El siguiente capítulo explica los pasos a seguir para el desarrollo de un Sistema Experto, estos pasos forman parte del método llamado "Rapid Prototyping" (construcción rápida de prototipos), el cual es considerado el método ideal para la construcción de Sistemas Expertos en comparación con el método del Ciclo de Vida de un sistema el cual es mejor utilizado cuando se construyen Sistemas Expertos Hibridos, es decir, aquellos sistemas donde se combina un Sistema Experto con un Sistema Convencional.

7.1 Desarrollo de un sistema experto.

Para desarrollar un Sistema Experto existen diferentes métodos los cuales pueden ser usados para determinar los requerimientos básicos y concluir con éxito la construcción del sistema. El método adecuado para el desarrollo de un Sistema Experto es el llamado "Rapid Prototyping" (construcción rápida de prototipos), el cual es un método iterativo que consta de los siguientes pasos:

- **Identificación.**

  El paso principal del trabajo recae en el ingeniero del conocimiento y el experto, ya que estas son las dos personas que se encargan de adquirir y transmitir el conocimiento del cual se nutren los Sistemas Expertos. El primer paso consiste en identificar aquellos problemas que implican un procedimiento de solución basado en el conocimiento, es decir, que los Sistemas Expertos deben ser aplicados donde por la complejidad del problema, su comportamiento sea dinámico, o por la explosión combinatoria no resulte posible o rentable una solución convencional mediante procesamiento de datos, por ejemplo los Sistemas Expertos pueden ser aplicados para el control del tráfico vehícular en una ciudad, donde a través de parámetros estadísticos sobre el número de vehículos que transitan y otros factores tales como las horas más transitadas y la dirección geográfica hacia donde tienden ir la mayoría puede generar distintas propuestas de solución para descongestionar las distintas calles y permitir así un paso fluido vehícular a las horas pico. Precisamente en la primera fase de un proyecto es de vital importancia determinar correctamente el ámbito estrechamente delimitado de trabajo.

  Una vez que se ha determinado el problema o los problemas que pueden ser resueltos por un Sistema Experto, es necesario identificar para que usuarios será creado el sistema y que resultados espera, es primordial tener en cuenta las ideas del usuario para obtener la aceptación y en consecuencia para el éxito del sistema.

Una vez determinado el dominio del problema, hay que "cebar" el Sistema Experto poco a poco con los conocimientos del experto, quien debe comprobar constantemente si su conocimiento ha sido transmitido de la forma más conveniente. El ingeniero del conocimiento es responsable de una implementación correcta, pero no de la exactitud del conocimiento. La responsabilidad de esta exactitud recae en el experto.

- **Conceptualización.**

  La adquisición de conocimiento comienza prácticamente con la conceptualización. En esta temprana fase los problemas se localizan en un principio en dos niveles los cuales se presentan a continuación:

  1. El equipo de ingenieros de conocimientos están muy poco familiarizado con la problemática especial y la terminología del dominio del problema.
  2. Los conceptos y métodos de trabajo de cada experto de los cuales se pretende adquirir el conocimiento pueden no llegar a coincidir muy bien. Es decir que el programa debe contener sólo en cantidades ínfimas un conocimiento de manual generalmente reconocidos, es decir, de libros, mientras su contenido principal se basa como es usual en procedimientos empíricos, en apreciaciones de expertos, que como es natural, oscilan mucho en su terminología y contenido.

  Después de reconocer los principales problemas que surgen, en esta fase de conceptualización se debe tratar de adquirir de los expertos los hechos o experiencias más relevantes, que relación existe entre estos hechos y la validez y disponibilidad de los mismos, es decir que se debe escoger de toda la información o hechos dados por los expertos aquella que es más relevante o de mayor utilidad para la construcción del programa y solución del problema, de igual manera se debe verificar la resolución entre estos datos para establecer su validez y disponibilidad. Una vez

determinada la información útil para el programa se deben determinar las estrategias de solución, es decir, si se van a utilizar predicados, reglas, etc. y la cantidad de secuencia de las soluciones.

- **Formalización e implementación.**

  En esta fase de formalización se debe hacer un esbozo de la base de conocimientos para poder determinar los requerimientos del mecanismo de control. La base del conocimiento es construida con los hechos y experiencias adquiridas en la fase de conceptualización. Estos hechos son organizados y clasificados de una forma lógica tal y como lo hace la mente de un niño que adquiere su conocimiento desde su nacimiento, es decir, conforme un niño va creciendo va ordenando sus ideas en la mente. Es así como la mente puede equiparase a una base de conocimiento, por ejemplo, una de las primeras informaciones que ingresan a la mente es la funcionalidad que tienen los miembros del cuerpo; otras es la que va adquiriendo a lo largo de su desarrollo tal como cuando aprende a distinguir los colores, los sonidos, a reconocer a las personas allegadas a éste, etc. Al igual que un ente individual introduce de diferentes formas los datos que adquiere durante su desarrollo, en un sistema experto la introducción de estos datos a su base de conocimientos depende de la metodología que utilice el lenguaje que se esta ocupando. Por ejemplo, en la base de conocimientos construida en el sistema experto para el diagnóstico de prótesis de miembros inferiores se uso el lenguaje de programación win-prolog, que utiliza el método de encadenamiento hacia atrás (backward chaining), para la introducción de la información por medio de la lógica de predicados. A pesar que cada lenguaje que se utilice posee sus propios métodos y características para la elaboración de la base del conocimiento, el ingeniero del conocimiento debe partir de la organización y clasificación de la información adquirida que el experto le proporciona o que él mismo ha inferido utilizando heurísticas. También se deben hacer los esbozos de las interfaces de usuarios y las del sistema para luego hacer una elección de la herramienta que se utilizará en la fase de implementación; dentro de

este conjunto de herramientas podemos mencionar los lenguajes de programación simbólicos (procesadores de símbolos) como Lisp, Prolog y demás lenguajes presentados anteriormente en este documento. Estos lenguajes son especialmente apropiados para la representación formalizada de conocimiento, son interpretativos (las órdenes son inmediatamente ejecutadas por el ordenador) y orientados al diálogo. Otras herramientas que se pueden utilizar son los entornos de desarrollo de programas, tal como win-prolog, interlisp, ops5 y otros, estos son construidos a partir de lenguajes de programación simbólicos y se sobreponen a ellos. Estos entornos suministran funciones y métodos auxiliares complejos, como por ejemplo un editor, un compilador o una ayuda de búsqueda de errores ("debuger"), orientados a la estructura.

Otras de las herramientas que facilitan el trabajo del ingeniero del conocimiento son los "Shells", ya que poseen todos los elementos necesarios excepto la base de conocimientos. Algunos de estos son sólo herramientas para la estructuración del conocimiento, es decir, programas para la representación de interrelaciones de conocimientos; otros "Shells" ponen a disposición del usuario uno o más mecanismos de representación del conocimiento y mecanismos de inferencia; por otra parte existen "Shells" que ofrecen posibilidades adicionales de control o apoyan la configuración de la interface. Los "Shell" contienen ya, por lo tanto, todos los componentes de un Sistema Experto, excepto la base de conocimientos. Por ejemplo, en el presente proyecto es utilizado el "Flex Shell", el cual posee las características mencionadas anteriormente y facilitó la creación del sistema ya que se concentraron los esfuerzos en la organización y clasificación de la información adquirida para conformar la base de conocimiento.

Luego de seleccionar la herramienta adecuada, se debe crear un modelo capaz de funcionar y luego adaptar a éste la herramienta, es decir, pasar a la etapa de implementación. En esta etapa se debe tener en cuenta que cada herramienta posee un conjunto de formalismos útiles para la representación del conocimiento que ofrece un apoyo eficiente a la estructuración del

conocimiento del saber. Entre estos formalismos se pueden mencionar las reglas de producción, redes semánticas, los frames, y los factores de certeza.

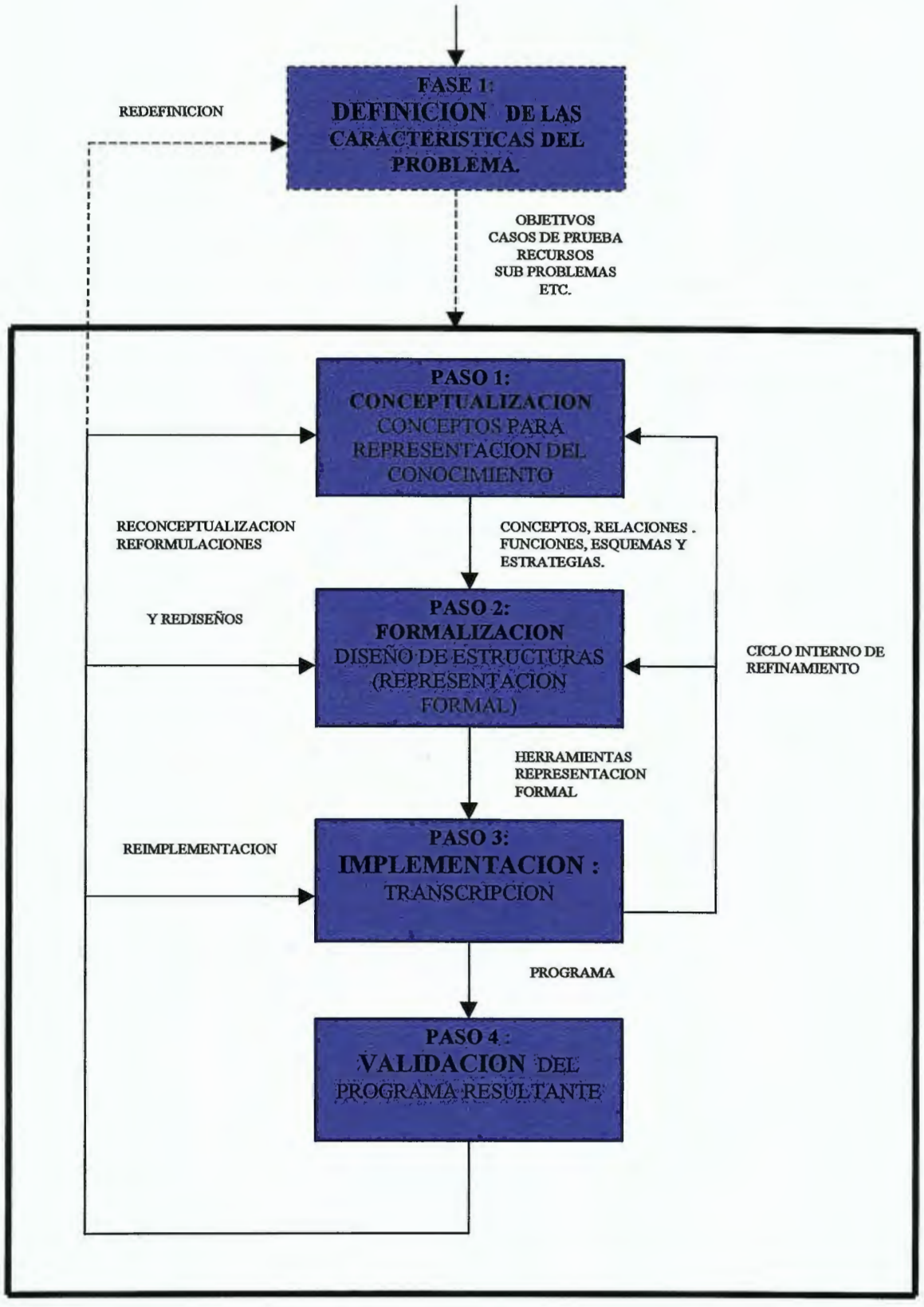
- **Verificación.**

  En esta fase se deben hacer una serie de pruebas al sistema y sobre la base de los resultados de estas pruebas se deben realizar los arreglos necesarios para que el sistema funcione de forma óptima, es decir que el sistema debe proporcionar soluciones correctas y precisas.

  En esta etapa se debe observar el comportamiento del sistema, determinar si las explicaciones son comprensibles y para ello se deben crear casos de test y ampliarlos y si es necesario se tiene que adaptar la herramienta seleccionada al ámbito del problema para obtener un sistema estable, útil y que cumpla con los requerimientos del producto.

Estos pasos son repetidos hasta el cumplimiento de un criterio de interrupción, es decir, hasta que se determina que el sistema trabaja de forma adecuada. De esta forma es posible crear un prototipo capaz de ejecutar una función y de mejorarla cada vez más con un esfuerzo de desarrollo relativamente pequeño.

PASOS PARA EL DESARROLLO DE LOS PROTOTIPOS



CAPITULO VIII
GUIA PARA INTERACTUAR CON EL EXPERTO

El objetivo de este capítulo es el de presentar una guía para el ingeniero del conocimiento, de tal forma que esta le ayude a interactuar con el experto a fin de extraer la información necesaria para el desarrollo de la base del conocimiento.

8.1 Interacción con el experto.

Para el desarrollo de la base del conocimiento, todo ingeniero del conocimiento debe buscar la información en diferentes fuentes del conocimiento: la literatura especializada, los expertos y los ejemplos. En algunos casos (sobre todo en los campos prácticos), no se puede codificar y registrar la experiencia de los expertos, por lo que se hace necesario de un experto viviente por la sencilla razón de que es poca la información disponible en forma escrita.

La mayor dificultad que se presenta en la adquisición del conocimiento es que el experto tal vez no sea capaz de verbalizar el conocimiento que utiliza. Con frecuencia, el experto ni siquiera está consciente de las reglas que sigue para resolver un problema; su conocimiento existe a nivel de subconsciente. Es por esto que sería conveniente que los expertos se utilizarán para escudriñar ejemplos, es decir, que a través de los ejemplos que cada experto va explicando, así se ira obteniendo la información más difícil de conseguir. Desafortunadamente no existe la ciencia de adquisición del conocimiento, es decir cómo interactuar con el experto.

Determinar la forma exacta de establecer una comunicación perfecta para la adquisición del conocimiento, tiende a ser "ad-hoc" y frecuentemente no es más que sentido común. Al parecer muchos de los métodos propuestos sólo funcionan con algunos sujetos o en algunos casos. Por lo antes expuesto, la presente guía da los lineamientos generales que todo ingeniero del conocimiento debe cumplir para adquirir el conocimiento de un experto o de un grupo de expertos sea cual fuere el área de aplicación del sistema.

Todo ingeniero del conocimiento debe ejecutar una seria de pasos necesarios para adquirir un mínimo de conocimiento necesario para completar la base del conocimiento del sistema experto que se pretende construir.

Para empezar el proceso de interacción con el experto todo ingeniero del conocimiento debe establecer en primer lugar una planificación y un horario. Las reuniones deben ser planificadas y organizadas para maximizar el acceso al experto y eliminar en lo posible las interrupciones, es decir, que se debe establecer un compromiso entre el ingeniero del conocimiento y el experto de tal forma que se obtenga el total apoyo y atención del último. También se debe planificar el acceso total a los instrumentos de implementación y se debe establecer el lugar de las reuniones, es aquí donde se debe decidir cual es el lugar idóneo para las reuniones o entrevistas: que puede ser el lugar habitual de trabajo del experto o el lugar de implementación del sistema.

Una vez concluida la planificación se realiza la primera reunión con el experto donde se discuten los puntos a tratar durante toda la etapa de creación del sistema. Es en esta fase donde el ingeniero del conocimiento debe pedirle al experto que lo familiarice con el dominio del problema y la terminología a utilizar, también se trata de obtener información para la primera base de conocimiento, tratando de preparar un documento que sirva de apoyo. Para obtener del experto sus conocimientos y experiencias se debe utilizar en un principio entrevista no estructuradas con las que se pretende ir obteniendo la información en forma general, y posteriormente organizarlo, verificarlo e implementarlo.

Luego de haber obtenido información general y haber conseguido que el ingeniero del conocimiento se haya familiarizado con el dominio, se debe buscar realizar una grabación lo mas detallada posible de lo que el experto hace en un día normal, pidiéndole que verbalice lo que va realizando, siempre que sea posible. En esta etapa se utilizan entrevistas estructuradas con las que se busca obtener respuestas especificas y que serán utilizadas como información para el buen funcionamiento del sistema.

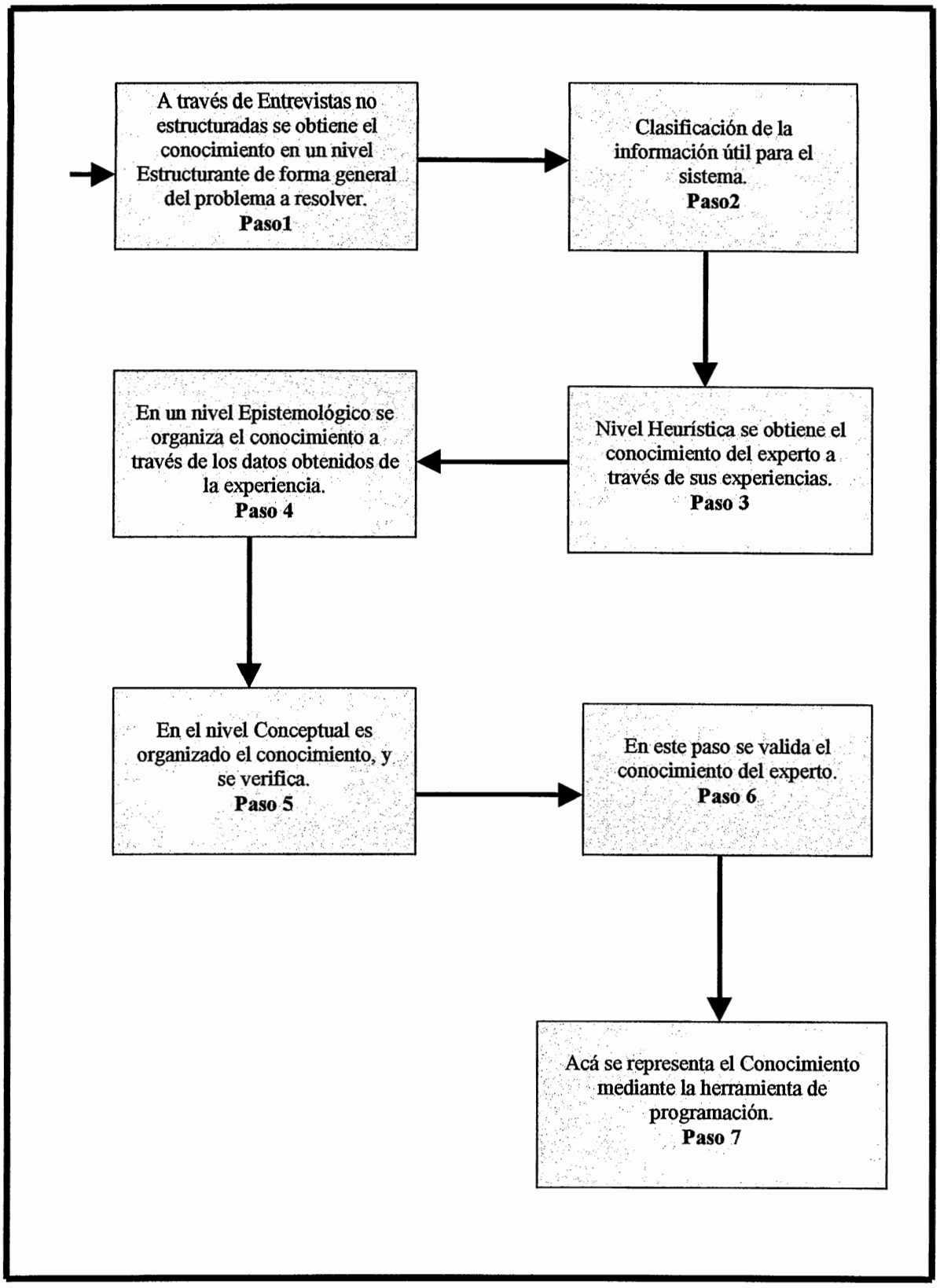
Como se expuso en la introducción de este capítulo, la mayor dificultad que se le presenta al experto es la de verbalizar el conocimiento que utiliza, es por eso que una de las técnicas eficaces para poder ayudar al experto y por consiguiente a uno mismo, es el uso de entrevistas estructuradas.

El uso de estas entrevistas estructuradas se dan una vez que el ingeniero del conocimiento se encuentra familiarizado con el dominio del problema y por lo tanto esta en la posición de saber que información es la que necesita el sistema para funcionar correctamente y así terminar de construir la base del conocimiento. El tipo de información que se busca con las entrevistas estructuradas son las estrategias del experto, es decir, que con el uso de preguntas específicas se pretende documentar los conocimientos de cómo el experto aborda el problema que se esta desarrollando, como éste trata de evitar las dificultades y que otros intentos hace el experto cuando no se puede seguir adelante.

Hasta este punto se espera haber comprendido el dominio del problema y luego haber establecido la información que se necesita para complementar la base del conocimiento con los conceptos y experiencias del experto, no debe olvidarse en ningún momento emplear una de las técnicas automáticas que existen para terminar de obtener la información, como lo es la inducción, la cual se obtiene al pedirle al experto que escudriñe unos cuantos ejemplos de tal forma que se logren identificar las posibles variantes del problema que se esta tratando de resolver, así como de las posible respuestas que existen a esas variantes.

Si se cumple con los pasos generales antes descritos y además utiliza su ingenio para adquirir el conocimiento del experto, es casi seguro que la base del conocimiento que se obtendrá tendrá toda la información necesaria para que el sistema en el cual se utilizará cumplirá con todas las expectativas para lo que será creado.

PASOS PARA OBTENER EL CONOCIMIENTO DEL EXPERTO



CAPITULO IX
ANALISIS Y DISEÑO

Este capítulo presenta el análisis creado para determinar la secuencia a seguir en la construcción del prototipo de un Sistema Experto para diagnóstico de prótesis de miembros inferiores. Durante esta etapa de análisis se detallan cada una de las fases realizadas comenzando con la fase de definición del problema a resolver y adquisición de datos, luego se realiza una segunda fase donde se especifica la forma de organización de la información, la cual es necesaria para poder llegar a un diagnóstico.

Con el uso de la información recolectada en la etapa del análisis, es posible elaborar el diseño gráfico del sistema a crear, el cual refleja la forma en que fluye la información para alcanzar una respuesta correcta.

9.1 Análisis del sistema.

Para el desarrollo del análisis del problema se han realizado las siguientes actividades que permiten mostrar la secuencia para la elaboración del prototipo:

1. La fase inicial ha consistido en la evaluación o definición del dominio del problema a investigar, el cual es la construcción de un prototipo para diagnóstico de prótesis del miembro inferior, que consta de un proceso de adquisición de datos del paciente tales como:

    - Nombres y Apellidos del paciente
    - Edad
    - Sexo
    - Lugar de donde proviene
    - Trabajo u oficio que desempeña
    - Si practica algún deporte
    - Si posee alguna enfermedad congénita
    - Si es alérgico a algo
    - Diagnóstico del doctor que lo remite
    - Tipo de amputación que posee

Después de haber obtenido datos generales del paciente se pasa a la etapa de la toma de medidas, pero esto no lo hará el programa, debido a que dicha actividad depende nada más del técnico que diseñará y elaborará la prótesis. La toma de medidas se lleva a cabo a través de varias evaluaciones, las cuales dependen de las fechas que el médico determine.

Como primera toma de medida se hace una evaluación del negativo el cual consiste en una medida hecha de yeso, y luego se realiza la evaluación del positivo la cual consiste en la prótesis

que va hacer utilizada por el paciente. Tanto la adquisición de la información del paciente, como la adquisición de las medidas que realiza el técnico ortopeda son los requerimientos básicos para diagnosticar un tipo de prótesis determinado.

2. Para la parte del análisis se organizan los datos adquiridos del paciente, ya que el diagnóstico de la prótesis depende mucho de las características físicas, entre las cuales se encuentran el sexo, la edad, tipo de amputación que posee y el estado de salud, es decir, si el paciente es alérgico a algún material, si es diabético o hemofílico, o físicamente se encuentra en condiciones de utilizar cualquier tipo de prótesis, dependiendo de todas estas características se diagnostica la prótesis a utilizar el paciente.

   Entre las características físicas que posee el paciente, las que se deben analizar más a fondo son las características que se derivan del tipo de amputación. Se deben tomar en cuenta, por ejemplo las ventajas o beneficios que se buscan obtener de una prótesis específica al saber que tipo de amputación posee, también es necesario conocer el nivel de amputación que el paciente tiene, ya que cada uno de estos niveles puede presentar diferentes problemas o traumas en el muñón. De igual forma se tiene que tomar en cuenta las diferentes característica de los tejidos que posee la piel de un paciente. El análisis exhaustivo de toda esta información es necesaria para pasar a la siguiente fase se realice un buen diagnostico y la prótesis sea la indicada.

3. Finalmente la última fase que se realiza, es el diagnóstico de la prótesis que el paciente va a utilizar, esta etapa depende de la organización de los datos y del análisis de ellos. Para poder iniciar el diagnóstico, es importante haber analizado perfectamente los datos, ya que a partir de ellos se genera la búsqueda de la mejor solución.

9.2 Diseño del sistema.

Para poder representar gráficamente el diseño del sistema se ha utilizado los Diagramas de Flujo de Datos (DFD) para obtener con más detalle las características de cada proceso. El primer diagrama que se presenta el diagrama de contexto del Sistema Experto, el cual representa los agentes externos que proporcionan la información necesaria de la cual se alimenta el sistema y el lugar donde almacena esta información. El segundo diagrama es un DFD del análisis y diagnóstico de prótesis para miembro inferior, con el cual se buscó describir en detalle el proceso de análisis y diagnóstico dentro del sistema, comenzando desde las entrevistas que el técnico ortopeda realiza a un paciente y de cuyas respuestas surge la información necesaria que es buscada o introducida en la base de conocimiento. Finalmente, el último gráfico representa la estructura de la base del conocimiento y la relación interna de su información.

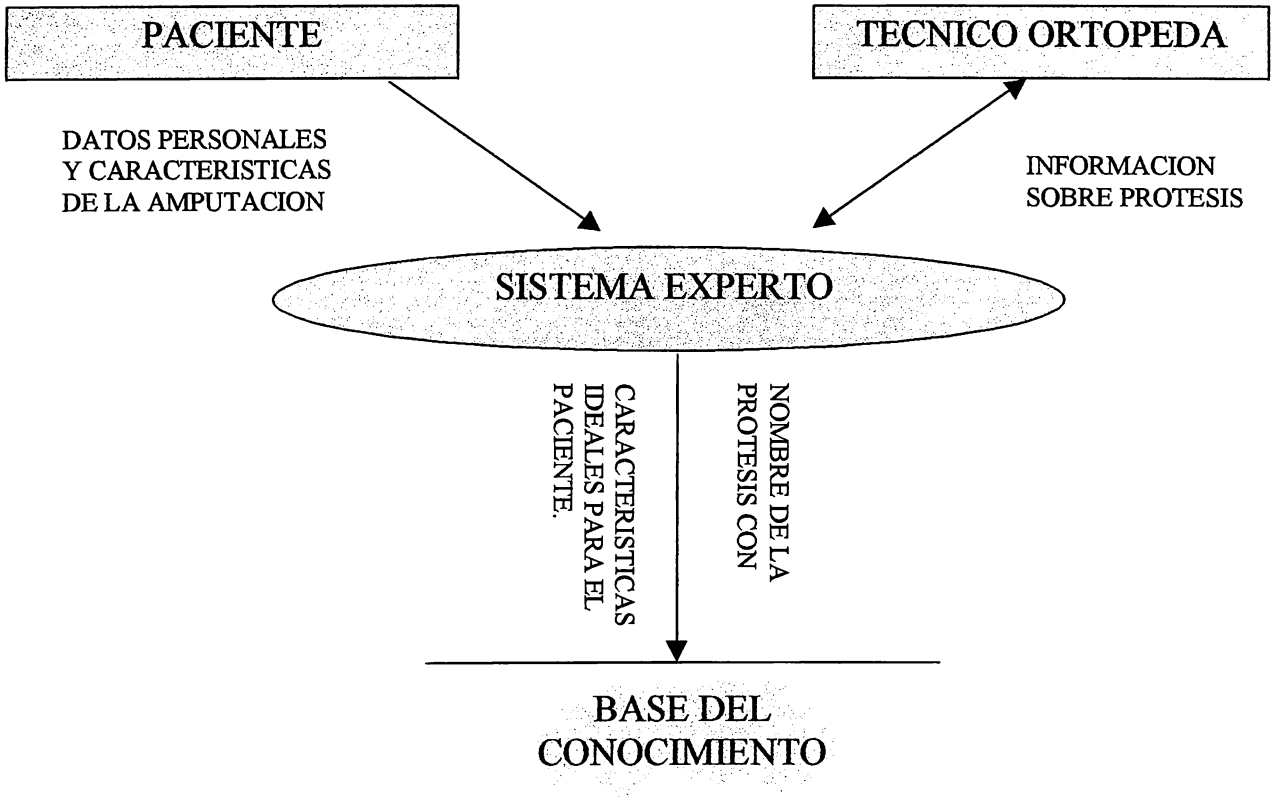
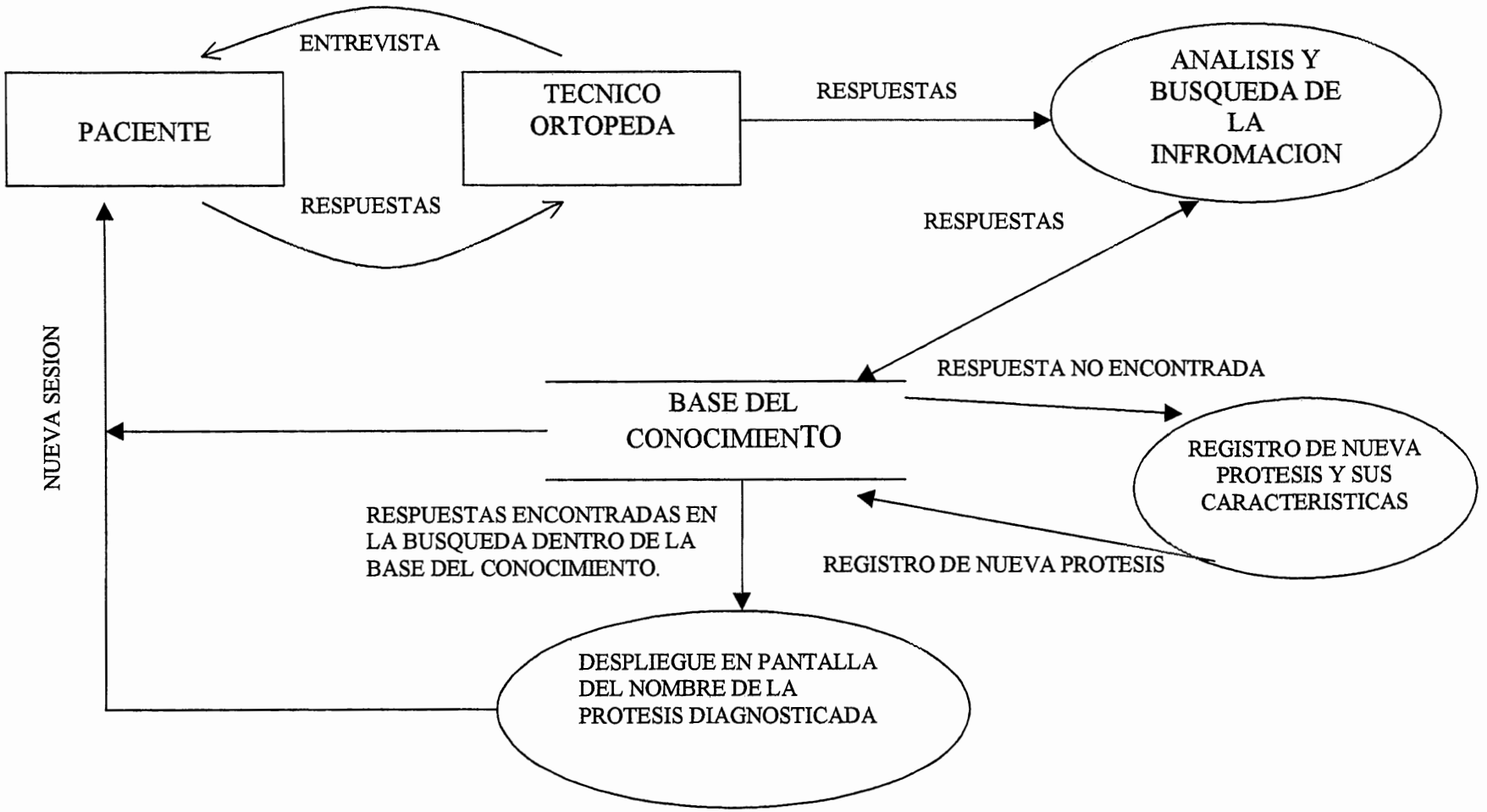
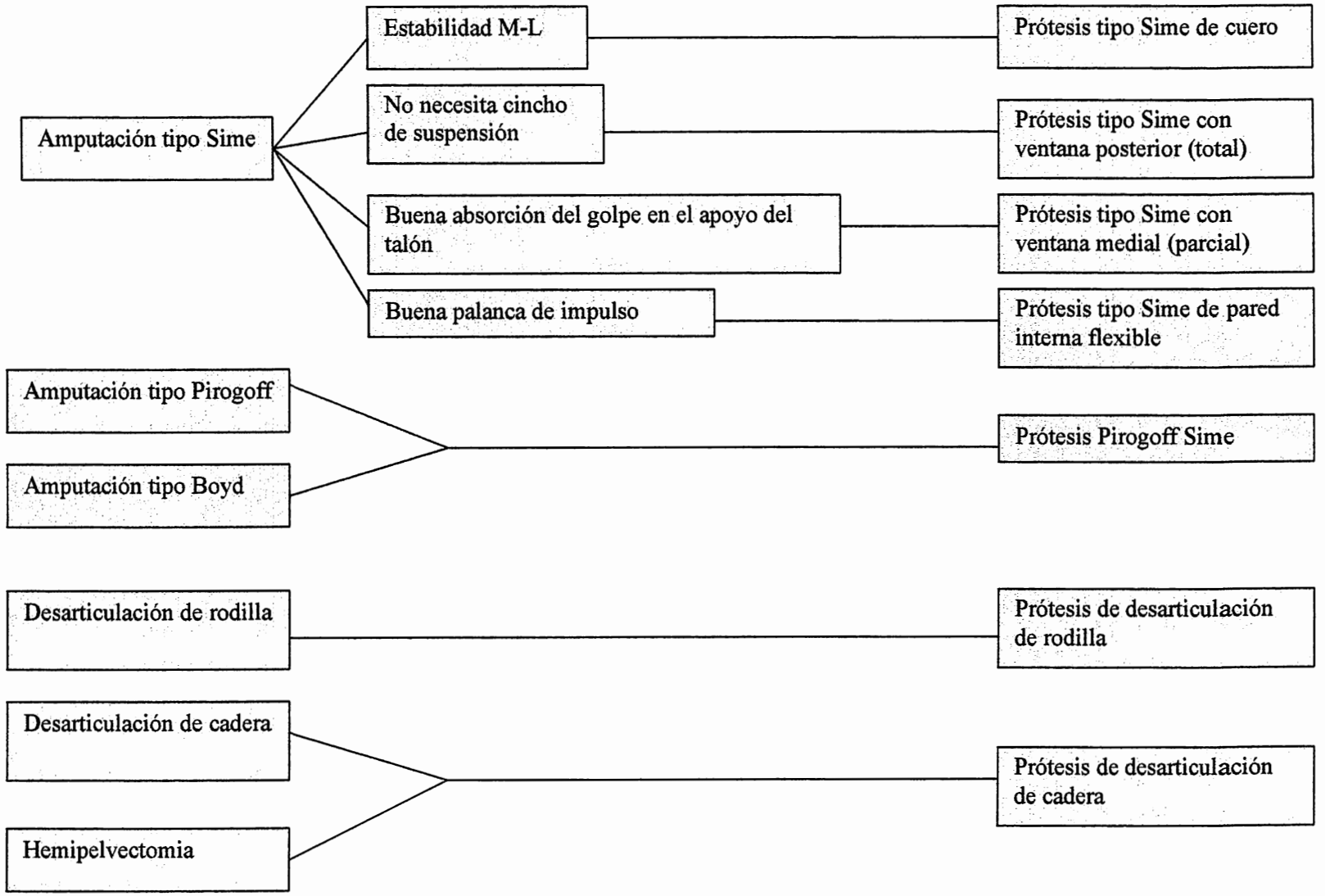


DIAGRAMA DE CONTEXTO PARA EL SISTEMA EXPERTO

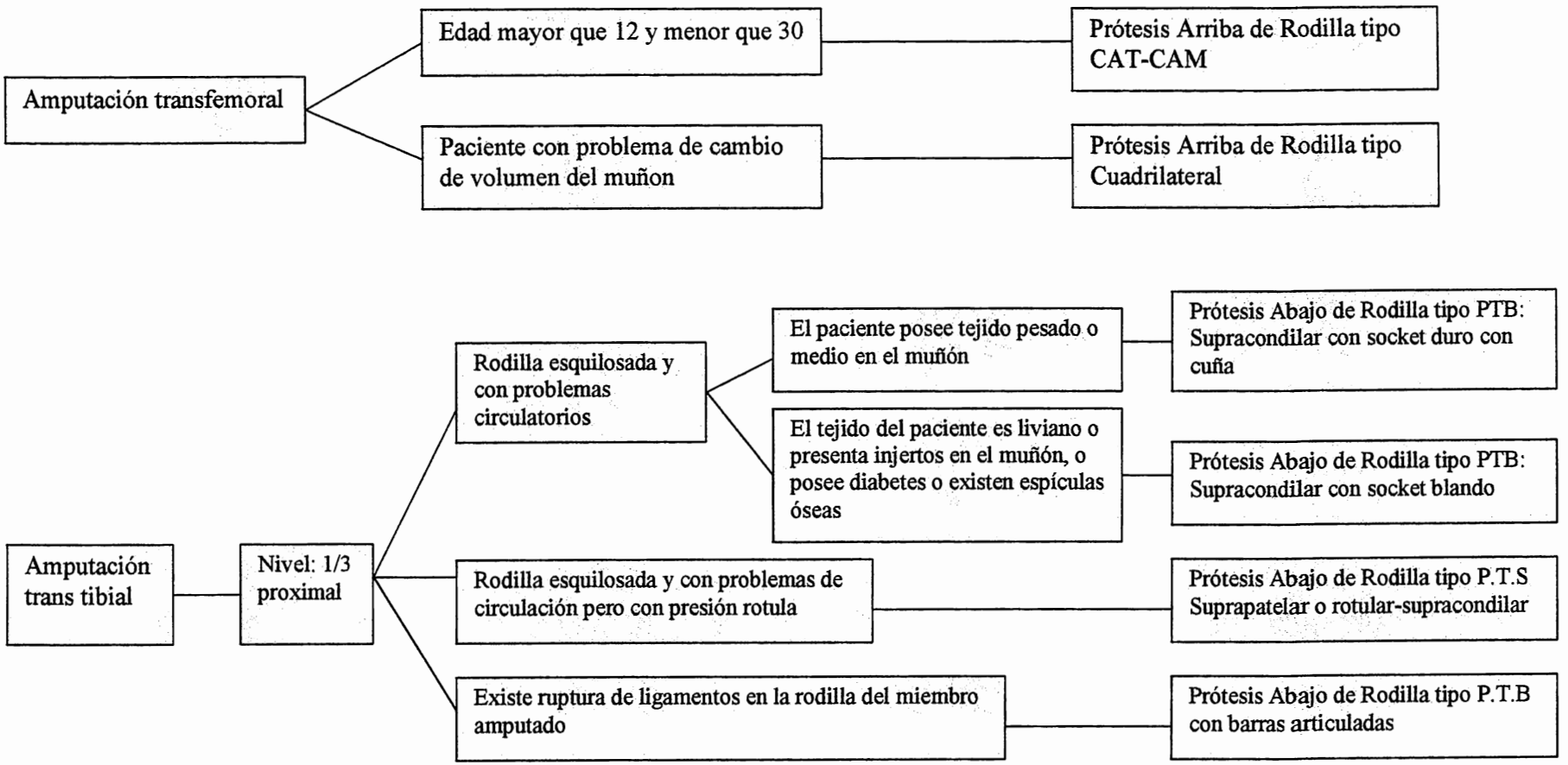


DFD PARA ANALISIS Y DIAGNOSTICO DE PROTESIS PARA MIEMBRO INFERIOR

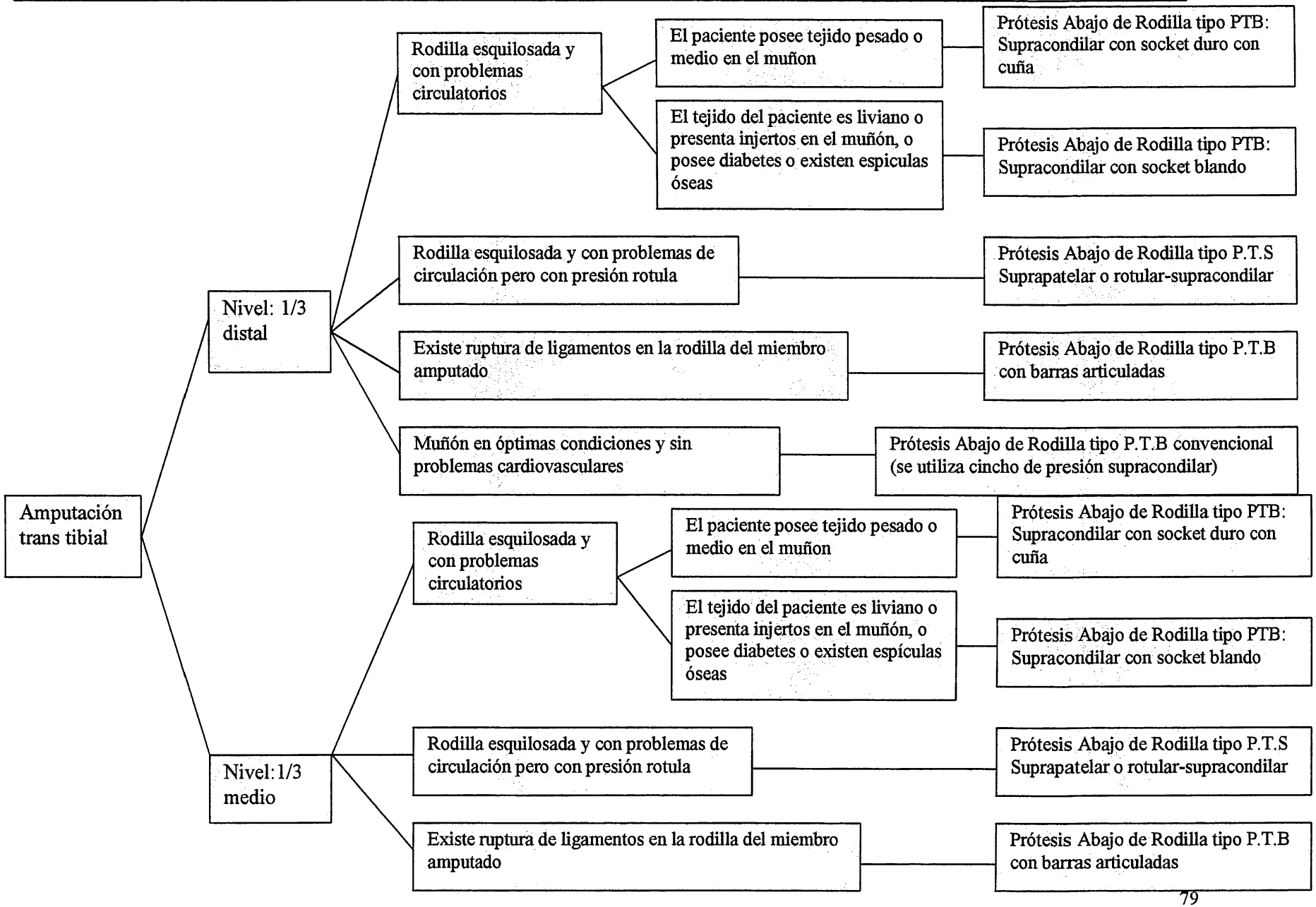
ESTRUCTURA Y RELACIÓN DE LA BASE DEL CONOCIMIENTO



CONTINUACION .....



CONTINUACION .....



CONCLUSIONES

Durante la elaboración de la investigación se concluyo sobre diferentes aspectos que son importantes en el desarrollo de un Sistema Experto:

- Para el desarrollo de Sistemas Expertos es necesario el conocimiento que proporciona el experto a través de sus hechos, experiencias, y sobre la base de ejemplos que posee. Además es importante enfatizar la necesidad de mantenerse en contacto a través de entrevistas para ir adquiriendo el conocimiento, que sirve para formar la base de conocimiento.
- Luego de extraer el conocimiento se hace necesario: organizarlo, codificarlo y validarlo. Después se realiza la representación del conocimiento a través de métodos formales como la lógica de predicados y la lógica modal.
- Antes de seleccionar un lenguaje de programación para Sistemas Expertos es necesario evaluar todas las características del problema que se esta tratando, para identificar cual de los diferentes lenguajes que se están investigando posee las herramientas adecuadas para la solución.
- Para la construcción del prototipo orientado al área de ortopedia, específicamente para el análisis y diagnóstico de prótesis para el miembro inferior se utilizó WIN-PROLOG, ya que se han tomado en cuenta los requerimientos básicos que debe tener un lenguaje de programación para Inteligencia Artificial. WIN-PROLOG permite fácilmente la construcción de un Sistema Experto, debido a que consta de las herramientas necesarias para la construcción de la base del conocimiento.

- Otro factor importante es la comunicación con el experto, debido a que es fundamental para que el prototipo pueda responder de la forma correcta, ya que el experto es la persona que posee los conocimientos necesarios para responder a las preguntas necesarias para el desarrollo de cualquier sistema.
- Las personas entrevistadas mostraron interés en el proyecto, porque piensan que este puede ser un prototipo de gran beneficio, por lo tanto cooperaron brindando toda la información pertinente o necesaria para la elaboración del prototipo.
- El "Datamite" es una herramienta externa de winprolog, el cual habilita reglas y conocimiento que puede ser descubierto en base de datos relacionales compatibles con ODBC. El "Datamite" no requiere experiencia especializada de programación, únicamente se apunta hacia un archivo, se le dice que salida es la que se espera y este descubrirá la data que se apega con los intereses que se tienen. La falta de esta herramienta dificultó la construcción del Sistema Experto debido a que se tuvo que entender la forma en que trabaja el Datamite para adicionar información a la base del conocimiento y para verificar la información ya existente y así convertirlo en un Sistema Experto inteligente.

RECOMENDACIONES

Para el desarrollo de Sistemas Expertos es necesario tener en cuenta las recomendaciones siguientes:

- La Inteligencia Artificial es campo de la computación que tiene que ser investigada más a fondo, específicamente el área de Sistemas Expertos, ya que por medio de estos se pueden resolver problemas tal como lo haría un humano, debido a que en estos sistemas se representa los conocimientos de un experto en base a sus experiencias, hechos, y ejemplos en una determinada área de aplicación.
- Es recomendable informar a los estudiantes en general, profesionales de la existencia de los Sistemas Expertos, ya que por medio de éstos se pueden resolver problemas complejos que se presenten en cualquier área, y para la solución de el los problemas existen diferentes lenguajes con características propias para ser utilizados en Inteligencia Artificial, como los descritos en el capitulo V de este documento.
- Otro factor importante es dar a conocer que para la adquisición del conocimiento del experto, se tiene que hacer uso de la técnica de la entrevista para extraer de esas personas sus experiencias, hechos, las entrevista en un inicio tienen que ser no estructuradas para conocer el dominio del problema de una forma general, luego realizar entrevistas estructuradas para conocer de una forma específica el problema que se esta tratando.
- La información que transmita el experto tiene que ser lo más exacta posible, debido a que esta información sirve para la construcción del Sistema Experto, ya que de él depende en gran medida que el sistema de respuestas correctas tal como lo haría el experto.

- Es recomendable utilizar la metodología de construcción rápida de prototipos "Rapid Prototyping", la cual consta de una serie de pasos iterativos, que se pueden utilizar para la solución de problemas en cualquier área de aplicación .
- Para cada área de aplicación de los Sistemas Expertos, el criterio para escoger la herramienta de programación a utilizar depende de que si los requerimientos del problema pueden ser cumplidos con las diferentes características de funcionalidad de la herramienta, por ejemplo si se tiene un problema cuyo requerimiento se adaptan más a un modelo de programación basada en eventos y con una interfaz bastante amigable y donde la creación de un motor inferencial complejo no se hace necesario, se podrá escoger un lenguaje visual como Visual Basic en lugar de Prolog.
- Se recomienda que, para facilitar la construcción de la base del conocimiento y volver el sistema experto aún más inteligente, se utilice una herramienta parecida al toolkit de winprolog llamado datamite, el cual puede establecer comunicación entre una base de datos relacional y el programa fuente a través de ODBC.

APENDICE A
ENTREVISTAS A EXPERTOS

El siguiente capítulo presenta las entrevistas realizadas a un grupo de expertos en el área del análisis y diagnóstico de prótesis para miembros inferiores, quienes trabajan en el proyecto ISRI (Instituto Salvadoreño de Rehabilitación de Inválidos) que se esta llevando a cabo en el ISRI (Instituto Salvadoreño de Rehabilitación de Inválidos) por la fundación alemana GTZ. Las siguientes entrevistas fueron utilizadas para adquirir el conocimiento del experto en el área de ortopedia, especialmente obtener información referente el diagnóstico de una prótesis del miembro inferior.

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

ESCUELA DE COMPUTACION

La presente entrevista tiene como objetivo adquirir el conocimiento del experto en el área de ortopedia, para la construcción de un prototipo orientado al área de ortopedia específicamente diagnóstico de prótesis miembro inferior.

La siguiente entrevista es realizada a la Dra. Canizalez, encargada de atender a los pacientes designada por GTZ.

1. ¿Que es una prótesis del miembro inferior?.

   R/ Una prótesis del miembro inferior se considera como el inicio de un nuevo proceso que, con la creación plástica de un órgano que es el muñón, con ayuda de un elemento externo protésico y con un tratamiento del proceso de protetización, intentará recuperar las funciones perdidas.

2. ¿Para que sirve una prótesis?.

   R/ La prótesis sirve para dar funcionalidad de algún miembro del cuerpo que haga falta.

3. ¿Cuales son las características que se le piden al paciente para obtener una prótesis?.

   R/ Nombres, apellidos, edad, estado de salud, sexo, educación, estado civil, trabajo u oficio, zona donde vive.

4. ¿En que causas existe una amputación?.

   Existe amputación por traumatismos, accidentes vasculares, tumores.

5. ¿Cuales son los factores que toma en cuenta el ortopedia para el diagnóstico de una prótesis?.

   R/ Los factores que se deben tomar en cuenta son la edad, sexo aunque influye poco en el proceso de protetización, lugar de residencia, trabajo, neuropatías ya que estas pueden dificultar o impedir la protetización por las alteraciones sensitivas superficiales y profundas.

   De estos factores los que incluye el Sistema Experto son edad, y las neuropatías.

6. ¿Que otros datos o factores influyen en el diagnóstico de la prótesis?.

   R/ Los factores que influyen son las cardiopatías e insuficiencia respiratoria, ya que la marcha con prótesis exige siempre un sobregasto enérgico, que estos pacientes pueden no estar capacitados para efectuar. Otro factor que influye es el estado de la otra extremidad inferior, ya que esta puede presentar signos de esquemia, rigideces articulares, atrofías musculares que produce.

7. ¿Las personas regresan a terapias después de la prótesis?.

   R/ La mayoría de los pacientes ya no regresan después de la prótesis.

8. ¿Después de la evaluación para el diagnóstico de la prótesis que se realiza?.

   R/ El paso siguiente es la toma de medidas, la cual la realiza el técnico protesista.

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

ESCUELA DE COMPUTACION

La presente entrevista tiene como objetivo adquirir el conocimiento del experto en el área de ortopedia, para la construcción de un prototipo orientado al área de ortopedia específicamente diagnóstico de prótesis miembro inferior.

Entrevista realizada al Ingeniero Ortopédico Carlos Matios, profesor de la Universidad Don Bosco y técnico protesista del proyecto de la GTZ.

1. ¿Cuales son los elementos de importancia que son necesarios valorar en el proceso de protetización?.

   R/ Los elementos importantes son la adecuación del muñón, y las condiciones generales del paciente. Ya que el muñón como órgano destinado a encajarse en la prótesis e impulsarla, debe reunir unas características específicas que lo califican como adecuado y estable. Entre las condiciones generales del paciente se encuentran las siguientes: una buena articulación arterial y venosa, que evite la esquemia o la estasís sanguínea, que la cicatriz sea la correcta y en el lugar adecuado.

2. ¿Que significa un muñón adecuado, y un muñón estable?.

R/ Muñón adecuado: No siempre el mejor muñón es el de mejor longitud. En ocasiones muñones más cortos obtienen después de la protetización resultados funcionales más satisfactorios que otros de nivel más distal.

La longitud adecuada esta entre 10 a 15 centímetros para la adaptación de una prótesis. Muñón adecuado es donde se puede adaptar una prótesis sin mayor problema.

Muñón estable: Significa que no hay problemas vasculares periféricos, esto quiere decir que tiene que haber una buena irrigación sanguínea en lo que es el muñón. Cuando se habla de irrigación sanguínea se dice que cumple con las amplitudes de movimiento.

3. ¿En todos los pacientes el muñón tiene que estar conformado de igual manera?.

   R/ No ya que el muñón varía un poco dependiendo del tipo de amputación.

4. ¿Cual es la longitud adecuada para cada tipo de muñón?.

   R/ La longitud del muñón depende de los criterios que el médico establece en el momento de realizar la cirugía. En casos traumáticos es mucho más relevante la vida del paciente que la longitud del muñón; cuando hay problemas por tumores óseos dependiendo que tan alto ha llegado el tumor así va a ser la medida que el médico designará.

   El técnico protesista adapta la prótesis a cada tipo de muñón independientemente de su longitud, forma o tamaño.

5. ¿Cuales son los parámetros del muñón para bajo rodilla y arriba rodilla?.

   R/ En bajo rodilla se habla de 10 a 15 centímetros y en arriba de rodilla se habla de 15 a 20 centimetros, este parámetro es para pacientes de estatura normal de 1.65 a 1.70 centímetros.

6. ¿Cambia el tipo de prótesis cuando una persona es diabética o hemofílica?.

   R/ El tipo de prótesis no cambia, lo que cambia son los tipos de materiales utilizados en la adaptación de la prótesis. Se pueden usar materiales suaves que permiten un menor grado de fricción, ya que en el diabético lo que se necesita evitar es cualquier tipo de ulceración o

laceración que pueda haber en el muñón, esto quiere decir que la prótesis se mantiene, lo que cambia son los materiales utilizados para adaptar la prótesis.

7. ¿Cuales son los materiales utilizados para los pacientes diabéticos?.

   R/ Pelite, es uno de los que más se utilizan, otro es la gelatina de silicon.

8. ¿Que pasa si el muñón esta bien y la cicatriz no esta correcta?.

   R/ Si la cicatriz no esta correcta, es decir, que no ha cerrado o hay algún punto que esta infectado, entonces no se pone prótesis inmediatamente, sino que se adecua la prótesis al tipo de muñon y cicatriz que presenta el paciente.

9. ¿Todas las prótesis aplican a cualquier edad que presenten los pacientes?.

   R/ Normalmente sí, para prótesis bajo rodillas desde los parciales de píe hasta el nivel de la rodilla aplica a cualquier edad; para las prótesis aplicadas arriba rodilla, o en desarticulaciones de caderas, aplican para cualquier edad, pero se tienen que ver los tipos de materiales, por ejemplo en un anciano se utilizan materiales más livianos.

10. ¿Hay diferencia entre una mujer y un hombre para indicar una prótesis?.

    R/ Diferencias no se establecen en la prótesis, pero hay diferencias anatómicas entre un hombre y una mujer, ya que las mujeres tienen mayor caderas, y se preocupan más por el aspecto cosmético.

11. ¿ Hay diferencias si práctica algún deporte, ejercicio o esfuerzo físico extraordinario para el diagnóstico de una prótesis?.

    R/ No hay diferencias, lo que cambia es el tipo de material, o los componentes que son utilizados para la prótesis.

12. ¿En casos de alergias que tipo de materiales utilizan para el diagnóstico de una prótesis?.

R/ Normalmente los materiales para fabricar las prótesis ya vienen con ciertas especificaciones antialergicas, si hay un caso especial, se determina mediante materiales que se utilizan en el campo, tales como termoplástico, resinas acrílicas.

13. ¿Dependen de la zona donde viven los materiales a utilizar en la prótesis?.

R/ Sí depende del lugar donde vive el paciente, así van hacer los materiales utilizados.

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

ESCUELA DE COMPUTACION

La presente entrevista tiene como objetivo adquirir el conocimiento del experto en el área de ortopedia, para la construcción de un prototipo orientado al área de ortopedia específicamente diagnóstico de prótesis miembro inferior.

Entrevista realizada Ingeniero Ortopédico Henner encargado del proyecto GTZ.

1. ¿Que es lo que tiene que hacer el paciente antes de una prótesis?.

   R/ El paciente tiene que ir a terapia física para que ponga en buenas condiciones el muñón, para mantener los arcos de movimiento, amplitudes articulares tales como la fuerza muscular, se tiene que eliminar sensibilidad del muñón.

2. ¿Que son amplitudes articulares?.

   R/ Amplitudes articulares es que la rodilla se pueda doblar, y así cualquier otra articulación como cadera, tobillo.

3. ¿Que es fuerza muscular?.

   R/ Fuerza muscular es mantener al paciente en forma, con ejercicios para que los músculos de las piernas estén bien.

4. ¿Tienen el mismo tipo de prótesis personas de escasos recursos?.

   R/ Es el mismo de tipo de prótesis para todas las personas.

5. ¿Explicar los pasos para diagnosticar una prótesis?.

R/ Consultar al médico para una evaluación médica. Posteriormente a eso, el trabajor social hace un estudio económico; luego llega al taller para que el técnico haga un examen funcional; después se pasa a la toma de medidas donde se obtiene el molde negativo, del molde negativo se obtiene un molde positivo donde se hacen todas las correcciones en base a las que se obtuvieron en la toma de medidas; y posteriormente se pasa a la laminación o plastificación del molde positivo, para que finalmente se haga un chequeo, donde se ve una prueba dinámica, y asi se pueda entregar la prótesis.

APENDICE B
CODIGO FUENTE DEL PROGRAMA

En el siguiente capítulo se presenta el código fuente utilizado para la creación del prototipo de Sistema Experto para el análisis y diagnóstico de prótesis del miembro inferior. Se utilizó la sintaxis del lenguaje Lpa-Win-prolog, conocida como PL. También se utilizó la sintaxis del flex shell de la compañía LPA, conocida como KSL; son dos programas con sintaxis similares que se intercambian parámetros para llamar sus funciones y el código es fácilmente leíble.

- Código fuente del programa escrito con la sintaxis de Flex Shell.

```
/* Esta parte contiene las preguntas del sistema y la función que llama a la base del conocimiento */

action experto;

        do repeat

        do restart

        and ask nombres

        and ask edad

        and indicación(Prótesis)

        and msgbox('respuesta',Protesis,48,Code)

        and adición(Prótesis)

        and close('sistemaexperto.pl')

        and compile('sistemaexperto.pl')

        and msgbox('Continuar','Desea Continuar?',36,Loop)

        until Loop is 7

        end repeat.

/* Acción que responde a la llamada hecha en el programa escrito en pl */

action intento(Result);

do a become Result

and catch(V,int(Res))

and relación(V,Res,Result).


action relación(V,Res,Result);

do if V is 0
```

```
then msgbox('Advertencia',Res,48,Codigo)
else ensure_loaded(sistemaexperto)
      and ventana2(Result)
end if.
/* Acción que adiciona la nueva información a la base del conocimiento */
action adición(Prótesis);
      do if Prótesis is  'El sistema no puede diagnosticar una prótesis con las indicaciones dadas,
debido a: no esta diseñado para diagnosticar ese tipo de prótesis o las indicaciones no son las
correctas para una prótesis existente'
          then ensure_loaded(sistemaexperto)
               and prolog(_)
      else msgbox('Experto','Diagnostico finalizado',48,Si)
      end if.
/* Esta parte contiene las preguntas del sistema y los grupos de posibles respuestas */
question indica1
      Cuál de las siguientes indicaciones posee el paciente?;
      choose one of grupo_indicaciones
      because Se necesita saber que indicación posee el paciente para
            determinar que prótesis necesita.
group grupo_indicaciones
'amputación tipo sime',  'amputación tipo pirogoff', 'amputación tipo Boyd', 'amputación trans tibial',
'desarticulación de Rodilla', 'amputación trans femoral', 'desarticulación de cadera',
''hemipelvectomia'.
```

question tipo

'Dependiendo de la ventaja que se busque que tenga la prótesis tipo Sime, así será el tipo de prótesis a construir.

Escoja cual de las siguientes ventajas busca en la prótesis?';

choose one of ventajasime

because La prótesis tipo Sime se caracteriza por recubrir un 100 por ciento la longitud original de la tibia y por formar una especie de bola en punto distal.

group ventajasime

'Estabilidad M-L', 'No necesita un cincho de suspensión', 'Buena absorción del golpe en el apoyo del talón', 'Buena palanca de impulso'.

question nivel

Que nivel de amputación posee el paciente?;

choose one of nivel

because Es necesario conocer que nivel de amputación se le ha hecho al paciente para diagnosticar el tipo de prótesis para amputaciones por debajo de rodilla necesita.

group nivel

'1/3 proximal', '1/3 medio', '1/3 distal'.

question indica2

Que tipo de indicación posee el paciente para este tipo de amputación?;

choose one of indica2

because Es necesario saber que tipo especifico de prótesis se le diagnosticara.

group indica2

'1) Rodilla esquilosada y con problemas circulatorios',

'2) Rodilla esquilosada y con problemas de circulación pero con presión rotular',

'3) Muñón en optimas condiciones y sin problemas cardiovasculares',

'4) Existe ruptura de ligamentos en la rodilla del miembro amputado'.

question estructura

Por favor escoja que características posee el paciente al que se le diagnosticara la prótesis;

choose one of estructura

because Es necesario estas características para determinar la estructura de la prótesis que sé esta analizando.

group estructura

'1) El paciente posee tejido pesado o medio en el muñón',

'2) El tejido del paciente es liviano o presenta injertos en el muñón, o posee diabetes o existen espiculas óseas'.

question pregunta

Tiene el paciente algún problema con cambio de volumen de muñón?;

choose one of grupo_pregunta.

group grupo_pregunta

'Si','No'.

question nombres

Por favor ingresa el nombre completo del paciente;

input set

because Es necesario llevar un registro de los datos personales del paciente.

question edad

Por favor ingrese la edad del paciente;

input integer

because La edad es necesaria para diagnosticar algunas de las prótesis.

/* Relación que verifica que una indicación no sea ingresada si ya existe */

relation int('No puede ingresar esa indicación, debido a que el sistema esta diseñado para diagnosticar todas las posibles prótesis con esa indicación')

if [a is 'amputación tipo sime' or a is 'amputación tipo pirogoff' or a is 'amputación tipo Boyd' or a is 'amputación trans tibial' or a is 'desarticulación de Rodilla' or a is 'amputación trans femoral' or a is 'desarticulación de cadera' or a is 'hemipelvectomia'].

% Relaciones del experto. Esta parte representa la base del conocimiento

relation indicación('prótesis tipo Sime de cuero')

if indica1 is 'amputación tipo sime'

and tipo is 'Estabilidad M-L'.

relation indicación('prótesis tipo Sime con ventana posterior (total)')

if indica1 is 'amputación tipo sime'

and tipo is 'No necesita cincho de suspensión'.

relation indicación('prótesis tipo Sime con ventana medial (parcial)')

if indica1 is 'amputación tipo sime'

and tipo is 'Buena absorción del golpe en el apoyo del talón'.

relation indicación('prótesis tipo Sime de pared interna flexible')

if indica1 is 'amputación tipo sime'

and tipo is 'Buena palanca de impulso'.

relation indicación('prótesis tipo Pirogoff Sime')

if indica1 is 'amputación tipo pirogoff'.

relation indicación('prótesis tipo Pirogoff Sime')

if indica1 is 'amputación tipo Boyd'.

relation indicación('prótesis Abajo de Rodilla tipo PTB: Supracondilar con socket duro con cu¤a')

if indica1 is 'amputación trans tibial'

and [nivel is '1/3 proximal' or nivel is'1/3 medio' or nivel is '1/3 distal']

and indica2 is '1) Rodilla esquilosada y con problemas circulatorios'

and [estructura is '1) El paciente posee tejido pesado o medio en el muñón'].

relation indicación('prótesis Abajo de Rodilla tipo PTB: Supracondilar con socket blando')

if indica1 is 'amputación trans tibial'

and [nivel is '1/3 proximal' or nivel is '1/3 medio' or nivel is '1/3 distal']

and indica2 is '1) Rodilla esquilosada y con problemas circulatorios'

and [estructura is '2) El tejido del paciente es liviano o presenta injertos en el muñón, o posee diabetes o existen espiculas óseas'].

relation indicación('Prótesis Abajo de Rodilla tipo P.T.S Suprapatelar o rotular-supracondilar')

if indica1 is 'amputación trans tibial'

and [nivel is '1/3 proximal' or nivel is '1/3 medio' or nivel is '1/3 distal']

and [indica2 is '2) Rodilla esquilosada y con problemas de circulación pero con presión rotular'].

relation indicación('Prótesis Abajo de Rodilla tipo P.T.B convencional (se utiliza cincho de presión supracondilar)')

if indica1 is 'amputación trans tibial'

and nivel is '1/3 medio'

and [indica2 is '3) Muñón en optimas condiciones y sin problemas cardiovasculares'].

relation indicación('Prótesis Abajo de Rodilla tipo P.T.B con barras articuladas')

if indica1 is 'amputación trans tibial'

and nivel is '1/3 proximal'

and [indica2 is '4) Existe ruptura de ligamentos en la rodilla del miembro amputado'].

relation indicación('El sistema no puede diagnosticar una prótesis con las indicaciones dadas, debido a: no esta dise¤ado para diagnosticar ese tipo de protesis o las indicaciones no son las correctas para una protesis existente')

if indica1 is 'amputación trans tibial'

and [nivel is '1/3 proximal' or nivel is '1/3 distal']

and [indica2 is '3) Muñón en optimas condiciones y sin problemas cardiovasculares'].

relation indicación('El sistema no puede diagnosticar una prótesis con las indicaciones dadas, debido a: no esta dise¤ado para diagnosticar ese tipo de prótesis o las indicaciones no son las correctas para una prótesis existente')

if indica1 is 'amputación trans tibial'

and nivel is '1/3 distal'

and [indica2 is '4) Existe ruptura de ligamentos en la rodilla del miembro amputado'].

relation indicación('prótesis de Desarticulación de Rodilla')

if indica1 is 'desarticulación de rodilla'.

relation indicación('prótesis Arriba de Rodilla tipo Cuadrilateral')

if indica1 is 'amputación trans femoral'

and pregunta is Si.

relation indicación('prótesis Arriba de Rodilla tipo CAT-CAM')

if indica1 is 'amputación trans femoral'

```
        and [edad is greater than 12 and edad is less than 30]
        and pregunta is No.
relation indicación('El paciente no tiene la edad adecuada para ocupar una prótesis tipo CAT-CAM,
por lo que debe utilizar una tipo Cuadrilateral')
        if indica1 is 'amputación trans femoral'
        and [edad is less than 13 or edad is greater than 31]
        and pregunta is No.
relation indicación('prótesis de desarticulación de cadera')
        if indica1 is 'desarticulación de cadera'.
relation indicación('prótesis de desarticulación de cadera')
        if indica1 is 'hemipelvectomia'.
```

- Código fuente escrito con la sintaxis de LPA-Win-Prolog.

```
% Función principal que es llamada desde el programa KSL.
prolog(_):-
msgbox('Creación de nueva prótesis: ', 'A continuacion se procedera a ingresar a la base de
conocimietos la nueva informacion. Favor responder las preguntas lo mas especifico
posible',48,Code),
        indicación,
        close('sistemaexperto.ksl'),
        compile('sistemaexperto.ksl').

%Manejadores de las ventanas.
amputacion_handler((amputacion,1),msg_button,_,Result):-
```

```
        wtext((amputacion,800),Res),
        atom_string(Result,Res).
relation_handler((relation,100),msg_button,_,Relation):-
        wtext((relation,800),Rela),
        atom_string(Relation,Rela).
/*Llamada a la ventana donde se pide el tipo de amputación para adicionar a la base del
conocimiento. */
indicación:-
        ventana1.
ventana1:-
        _S1 = [dlg_ownedbydesktop,ws_sysmenu,ws_caption],
        _S2 = [ws_child,ws_visible,ss_left],
        _S3=[ws_child,ws_visible,ws_tabstop,ws_border,es_left,es_multiline,es_autohscroll,es_auto
vscroll],
        _S4 = [ws_child,ws_visible,ws_tabstop,bs_pushbutton],
wdcreate(amputación, `tipo de amputacion`,235, 43, 372, 261, _S1 ),
        wccreate((amputacion,1000), static, `Por favor ingresa el tipo de amputación que posee tu
paciente. Por ejemplo: amputación tipo sime`,40,  40, 310,40, _S2 ),
        wccreate((amputacion,800),  edit,``,40, 90, 300,  80, _S3 ),
        wccreate((amputacion,1),button, `&Aceptar`, 140, 190,  90,  30, _S4 ),
        window_handler(amputacion,amputacion_handler),
       call_dialog(amputacion,Result),
       ensure_loaded('sistemaexperto.ksl'),
       intento(Result).
```

```
/* Segunda ventana que pide que ingrese el nombre de la nueva prótesis que el programa podrá
diagnosticar, una vez se ha verificado que la indicación para esta no existe en la base del
conocimiento. */
ventana2(Result):-
        _S1 = [dlg_ownedbydesktop,ws_sysmenu,ws_caption],
        _S2=[ws_child,ws_visible,ws_tabstop,ws_border,es_left,es_multiline,es_autovscroll,es_auto
hscroll],
        _S3 = [ws_child,ws_visible,ws_tabstop,bs_pushbutton],
        _S4 = [ws_child,ws_visible,ss_left],
wdcreate(relation, `Nombre de protesis`,235,43,372,261,_S1),
        wccreate((relation,800),edit,``,40,90,300,80,_S2),
        wccreate((relation,100),button,`&Aceptar`,140,190,90,30,_S3),
wccreate((relation,1000),static, `ingresa el nombre de la nueva prótesis que podré diagnosticar de
hoy en adelante.`,40,20,310,40,_S4),
        window_handler(relation,relation_handler),
        call_dialog(relation,Relation),
        write('~M~J')~>'sistemaexperto.ksl',
        write('relation indicación('')~>'sistemaexperto.ksl',
        write(Relation)~>'sistemaexperto.ksl',
        write('')~M~J')~>'sistemaexperto.ksl',
        write('if indica1 is '')~>'sistemaexperto.ksl',
        write(Result)~>'sistemaexperto.ksl',
        write(''.')~>'sistemaexperto.ksl',
```

```
write('~M~J')~>'sistemaexperto.ksl',

write('~M~J')~>'sistemaexperto.ksl',

len(Result,L),

see('puntero'),

fread(i,0,0,I),

seen,

A is I+L+1,

tell('sistemaexperto.ksl'),

outpos(I),

write(','"),

write(Result),

write('".'),

told,

tell('puntero'),

outpos(0),

fwrite(i,0,0,A),

told,

close('puntero'),

see('puntero1'),

fread(i,0,0,J),

seen,

tell('sistemaexperto.ksl'),

B is L+J,

outpos(J),
```

```
write(' or a is '"),
write(Result),
write('"].'),
told,
tell('puntero1'),
outpos(0),
fwrite(i,0,0,B),
told,
close('puntero1').
```

APENCIDE C

MANUAL DEL USUARIO DEL PROTOTIPO DE SISTEMA EXPERTO

Prototipo de Sistema Experto para el análisis y diagnóstico de Prótesis del Miembro Inferior

Versión 1.0

Manual del Usuario

UNIVERSIDAD DON BOSCO

MARZO 1999

Introducción

El presente manual forma parte de la documentación dirigida al técnico ortopeda encargado de la administración del Sistema Experto para el análisis y diagnostico de Prótesis del Miembro Inferior. Previo a la descripción de dicho sistema se brinda en la sección I una explicación del ambiente de trabajo del mismo y de su menú principal.

Requerimientos previos de la audiencia

Como condición previa a la lectura de este manual, se asume que el usuario tiene experiencia trabajando en ambientes Windows, ya sea con Windows NT o Windows 95 (todas marcas registradas de Microsoft Corp.). Además se asume que se tiene previamente instalado Win-Prolog 2.6 versión del programador y flex shell toolkit (ambas marcas registradas por LPA). Por lo tanto, la funcionalidad del sistema asociadas con las características estándar del sistema operativo no son descritas.

¿ Cuál es el contenido de este manual?

El manual del usuario se encuentra organizado en varias secciones:

Sección I, "Conocimientos Básicos sobre el Sistema", brinda una descripción de los puntos importantes que se deben conocer y aplicar para empezar con la utilización del prototipo de Sistema Experto para el análisis y diagnostico de prótesis del miembro inferior.

Sección II, "Ingresando al Sistema", explica la forma de ingresar al Prototipo de Sistema Experto para el análisis y diagnostico de prótesis del miembro inferior.

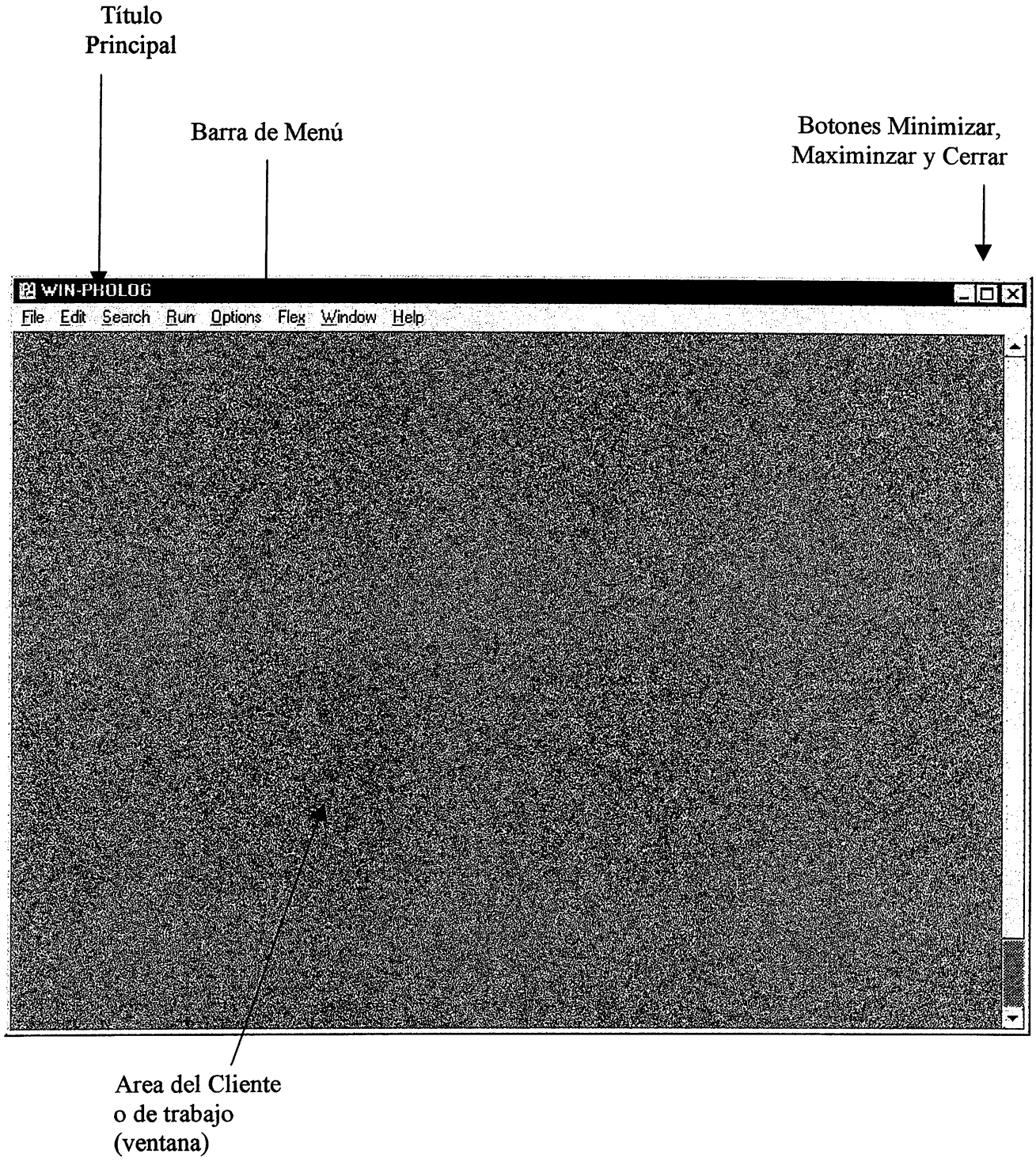
Sección III, "Análisis y Diagnostico de Prótesis", explica en la forma de introducir y escoger la información que el sistema requiere para dar como resultado un tipo de prótesis especifico.

Sección IV, "Ingreso de un nuevo tipo de Prótesis al sistema", detalla las razones y los pasos a seguir para ingresar un nuevo tipo de prótesis al sistema.

Sección V "Resolución de problemas", explica los posibles problemas que pueden surgir y la forma de cómo solucionarlos.

SECCIÓN I. CONOCIMIENTOS BÁSICOS SOBRE EL SISTEMA

El ambiente de trabajo para el usuario está constituido por una ventana de Windows como la que se muestra en la figura siguiente:



Como se muestra en la pantalla anterior, una ventana de Win-Prolog se divide en varias partes. Se tiene un título principal, una barra de menú y un título de ventana.

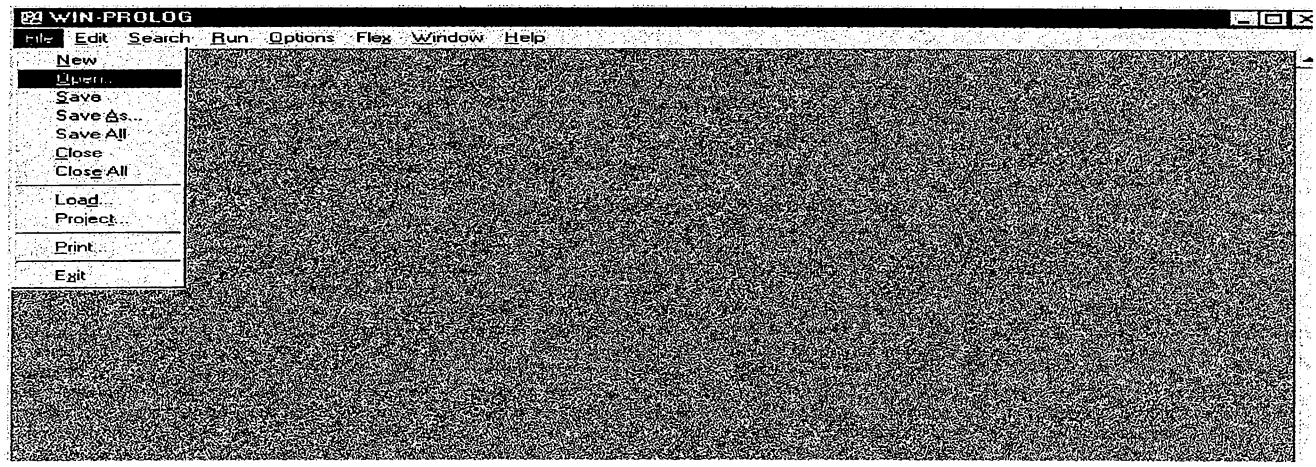
El **título principal** de la ventana contiene el nombre del programa.

La **barra de menú** implementa un menú pull-down de Windows (éste manual asume conocimientos básicos de la interface de Windows).

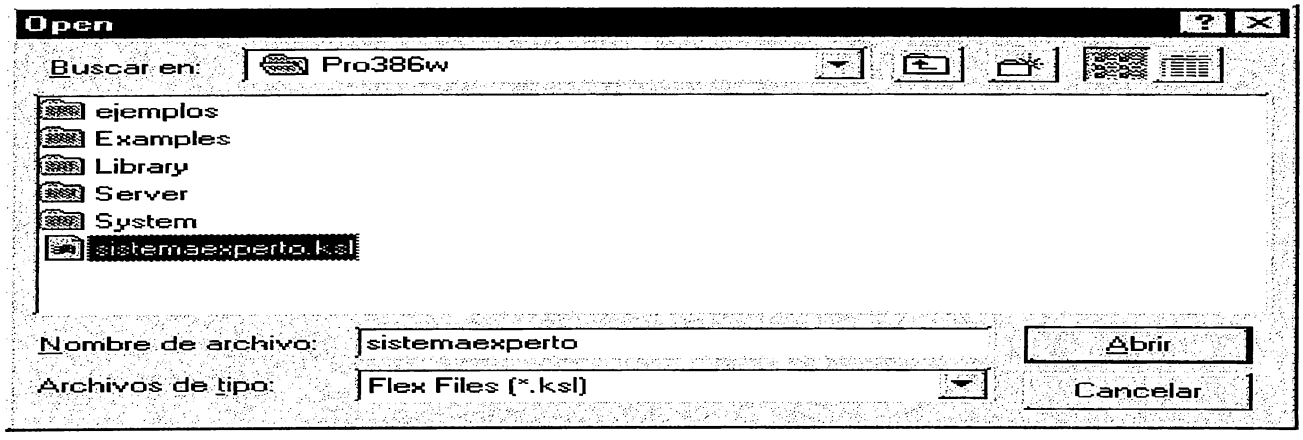
La **ventana o área del cliente** es un área donde se ingresan, modifican, eliminan o consultan datos. Es el área de trabajo del usuario.

SECCIÓN II. INGRESANDO AL SISTEMA

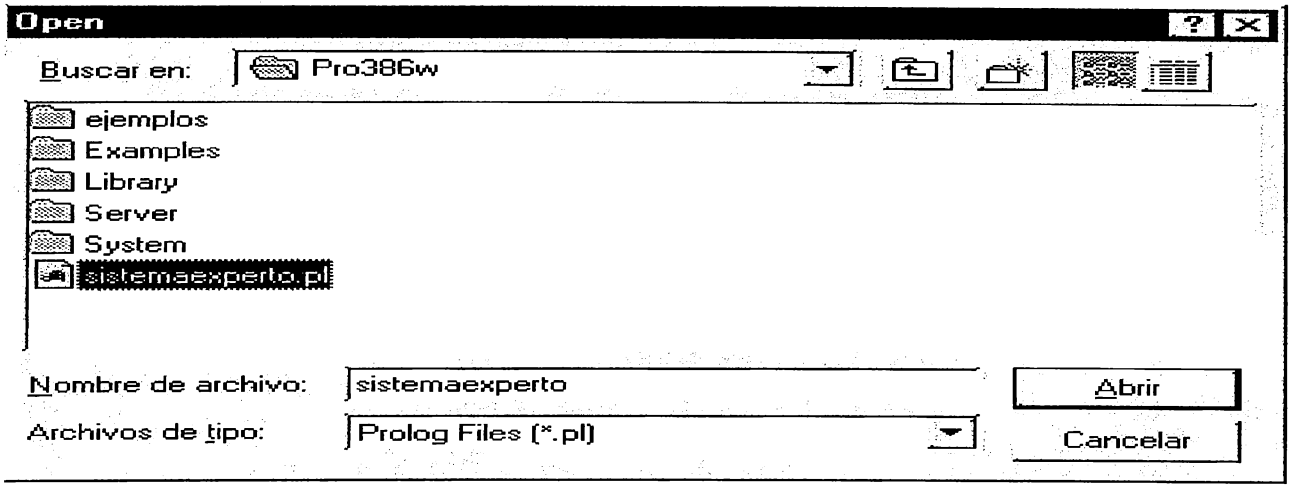
Desde el ambiente Windows, ejecutando el programa flex shell de LPA, usted iniciará la ejecución del shell tal y como se muestra en la figura anterior en la sección I. Después tendrá que presionar la opción File de la barra de menú de la ventana principal del flex shell de win-prolog y a continuación tendrá que hacer click sobre la opción Open como se muestra en la siguiente figura:



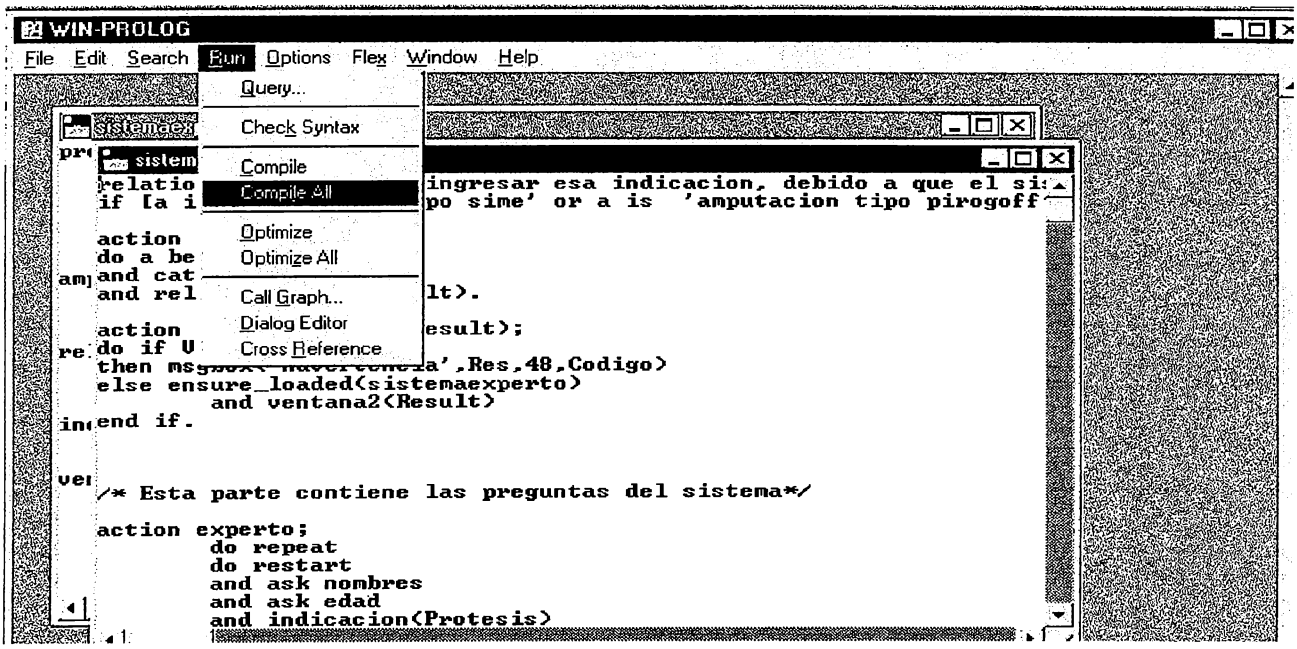
Luego de haber hecho click sobre Open, aparece la ventana para abrir archivos de la cual se debe buscar en el directorio donde haya copiado los archivos del Sistema Experto (en la siguiente figura es pro386w), a continuación escoger en la opción tipo de archivo los que tengan extensión KSL, es decir Flex Files y escoger el archivo llamado sistemaexperto y finalmente hacer click en abrir tal y como se muestra en la siguiente figura:



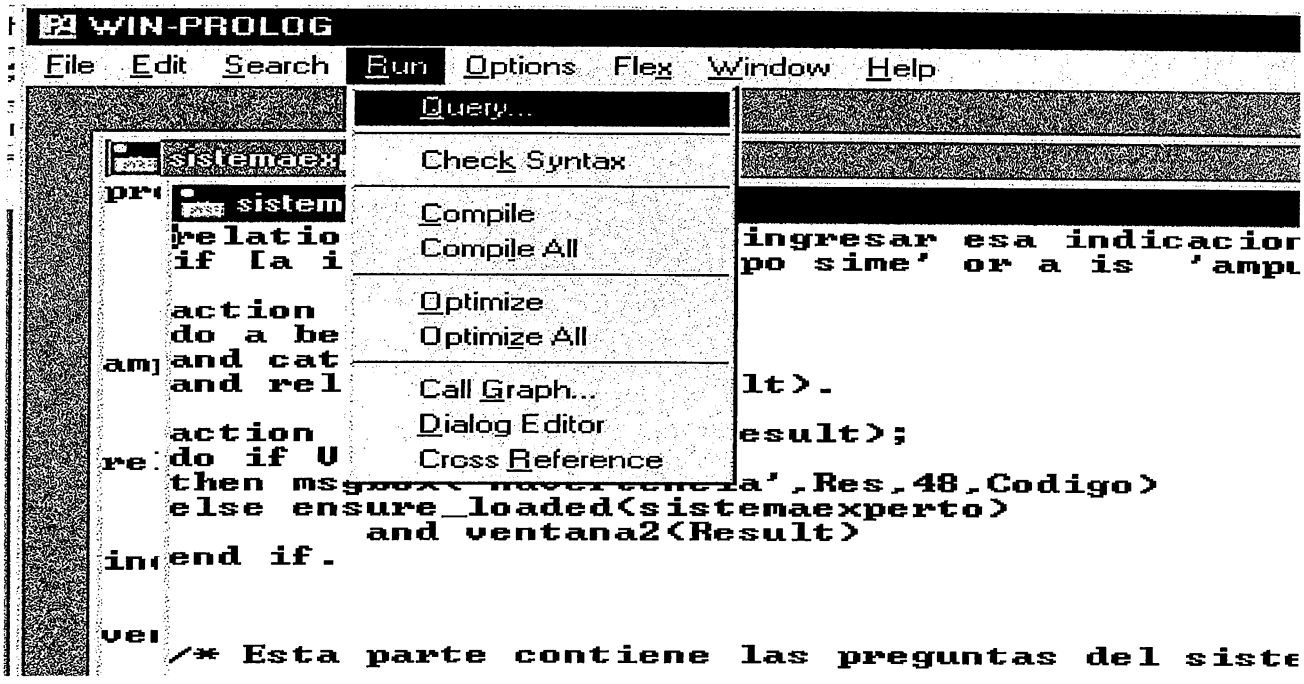
Una vez se ha realizado lo anterior, se debe volver a escoger la opción Open que se encuentra dentro de el menú File tal y como se explicó previamente. Esta vez se deberá escoger los archivos de tipo PL , es decir , Prolog Files, y el archivo con el nombre de sistemaexperto como se muestra en la siguiente figura:



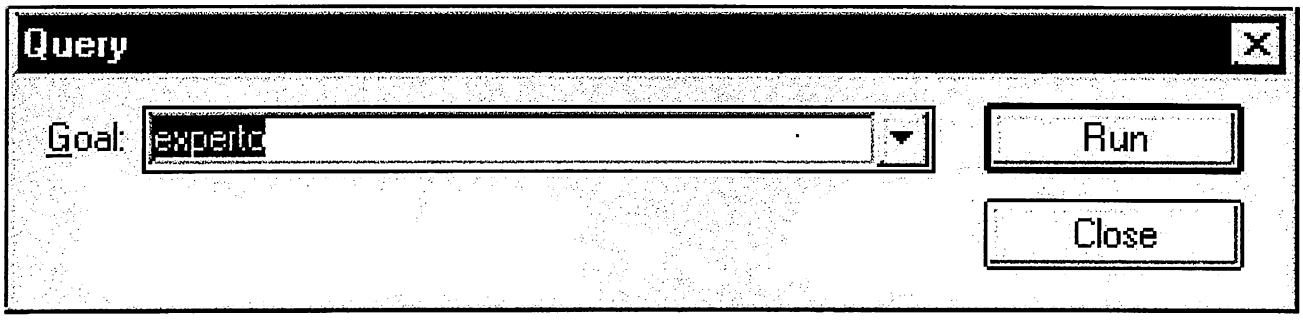
Una vez se han abierto los dos archivos que contienen el código fuente del sistema, se deben compilar para esto es necesario hacer click sobre la opción de la barra del menu llamada Run y luego escoger Compile All. Una representación gráfica del proceso anterior es el siguiente:



Una vez que se han compilado estos archivos se procede a correr el sistema. Para esto es necesario hacer click sobre Run en la barra de menú y escoger el item Query. En la siguiente figura se presenta las opciones que se describen anteriormente para abrir el cuadro de dialogo en el cual se ingresa el comando para ingresar al Sistema Experto:



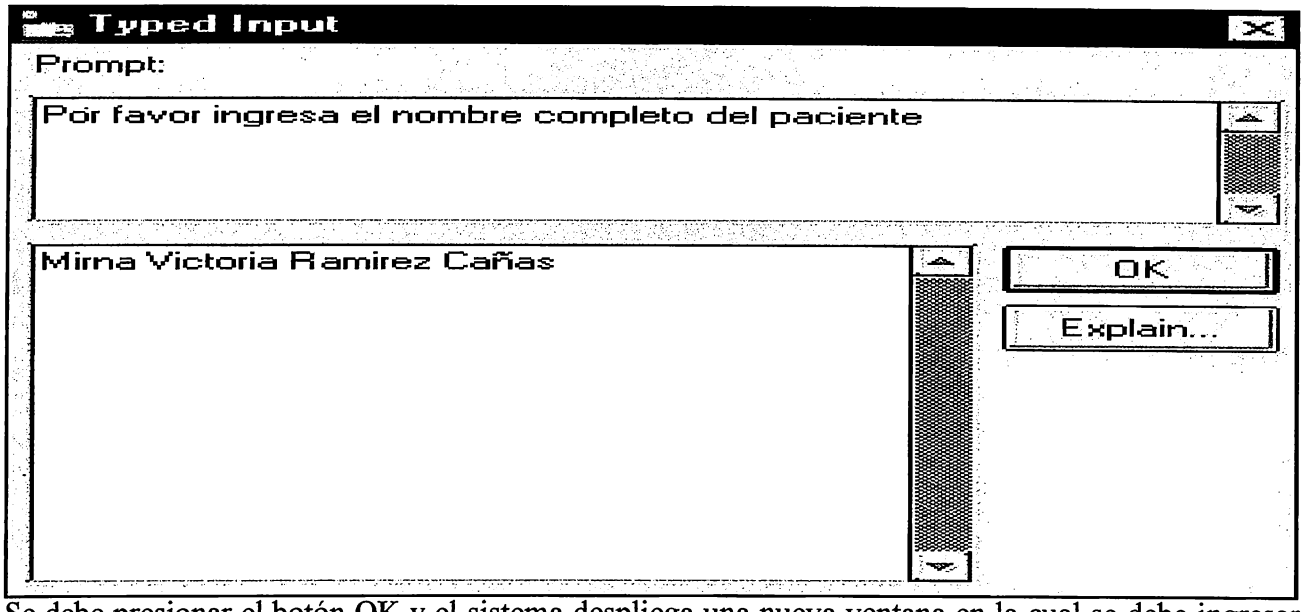
Luego de haber presionado Query se muestra la ventana llamada Query, en la cual se ingresa el comando "experto" (exclusivamente en minusculas) para comenzar a ejecutar el sistema. Este comando se ingresa en la casilla llamada Goal y luego se hace click sobre Run.



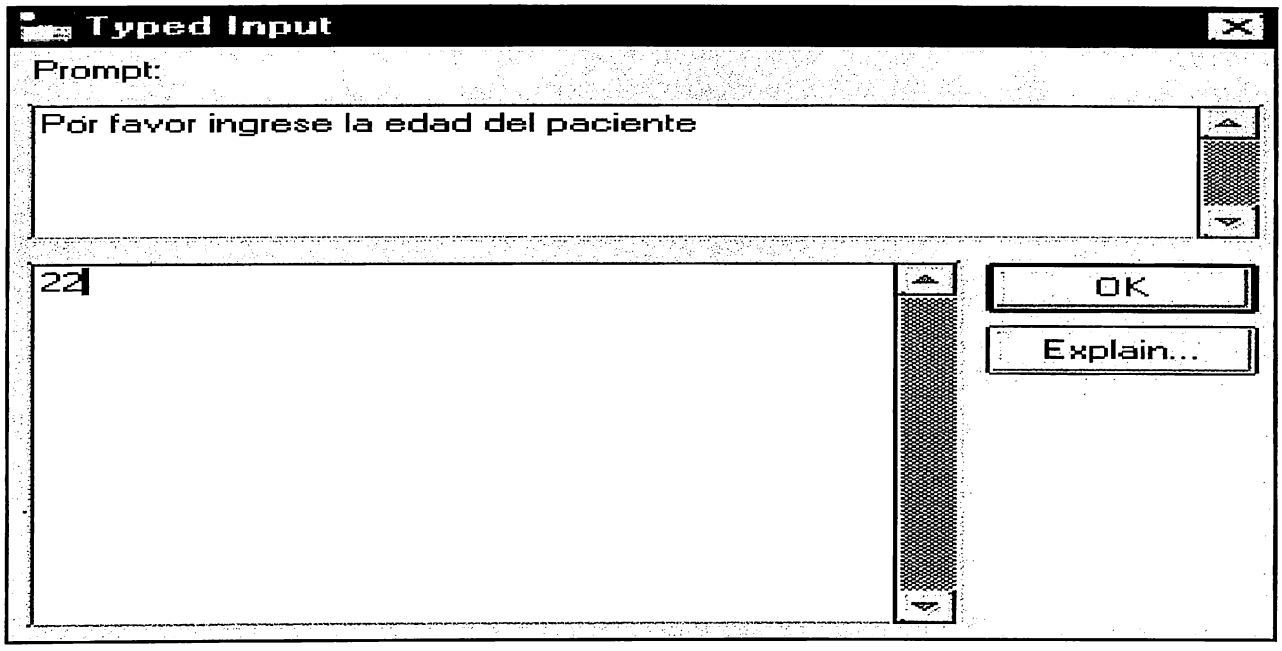
En caso de no ingresar el comando como es debido el sistema no responderá y por tanto no realizara acción alguna.

SECCIÓN III. ANÁLISIS Y DIAGNOSTICO DE PRÓTESIS

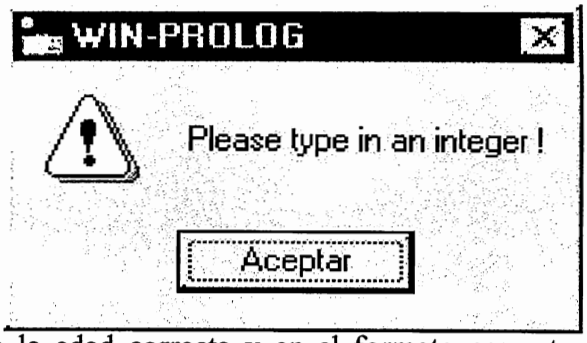
Una vez que se ha presionado el botón Run de la ventana Query, el sistema comienza a ejecutarse y se muestra la ventana llamada Typed Input, en la cual se le pide que ingrese los nombres del paciente:



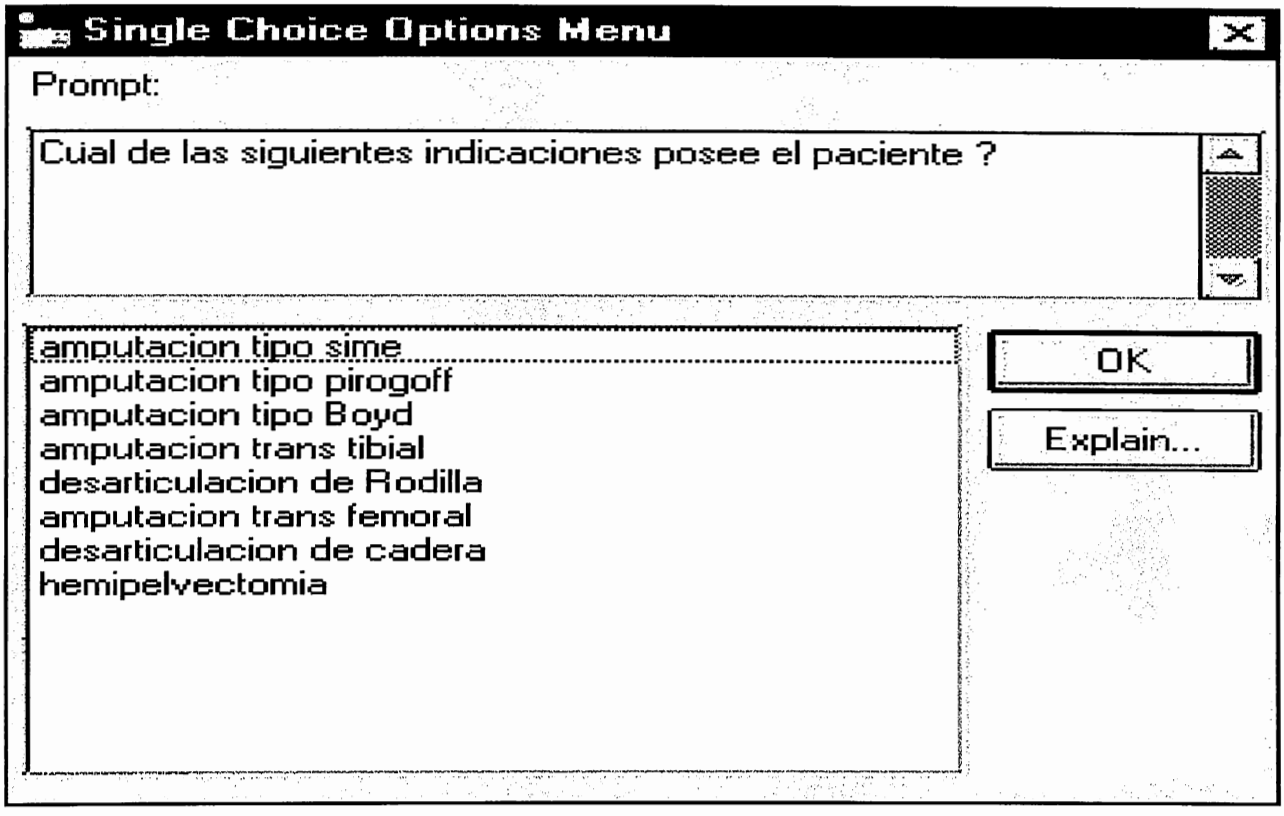
Se debe presionar el botón OK y el sistema despliega una nueva ventana en la cual se debe ingresar la edad del paciente:



Esta ventana únicamente aceptara que el usuario ingrese valores enteros de otra forma mostrará un mensaje en el cual explica que debe ingresar un entero:

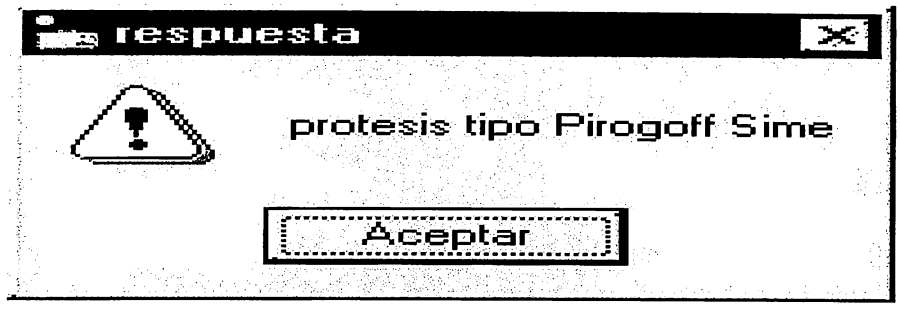


Una vez se ha ingresado la edad correcta y en el formato correcta, aparece la siguiente ventana llamada Single Choice Options Menu, en la cual se le pide al usuario que escoja una de las indicaciones para poder diagnosticar un tipo de prótesis especifico.



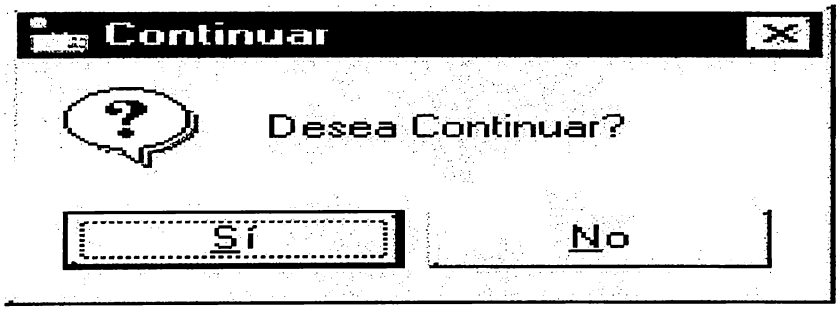
Al escoger un tipo de amputación podrían aparecer otras ventanas similares, es decir, el sistema pide al usuario que escoja la(s) característica(s) que posee el paciente.

Luego que se han contestado todas las preguntas, las respuestas son analizadas por el sistema y verificadas en la base del conocimiento para que posteriormente diagnostique un tipo de prótesis especifico, tal y como se muestra en la siguiente figura:



Una vez que se presione Aceptar el sistema presentará una ventana en la cual rectifica que el diagnostico a finalizado:

A continuación se le pregunta al usuario si desea realizar un nuevo diagnostico a través de la siguiente ventana:

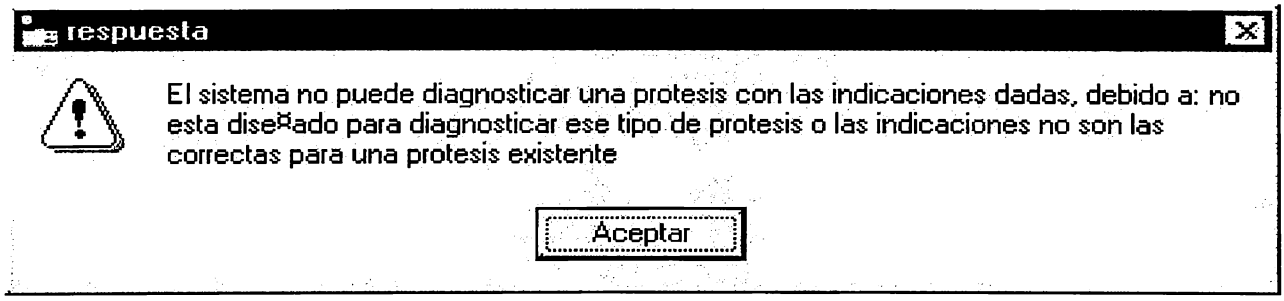


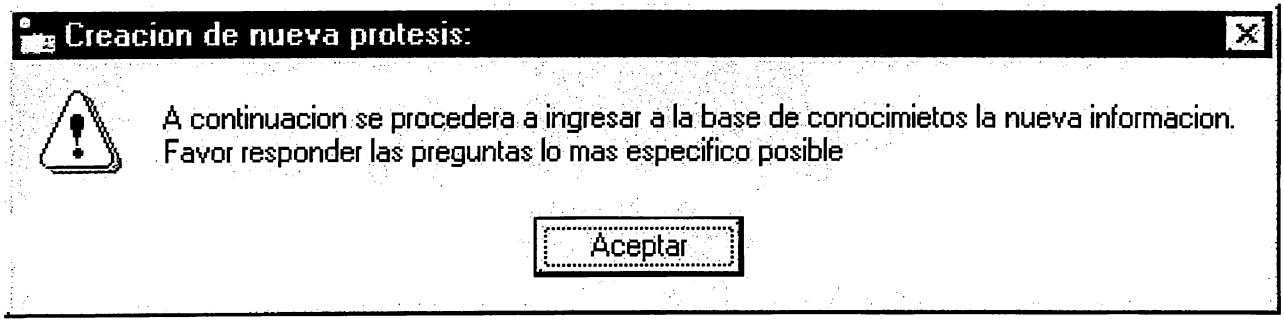
Si desea realizar otra consulta, presione continuar, en caso contrario presione que No.

SECCION IV. INGRESO DE UN NUEVO TIPO DE PROTESIS AL SISTEMA.

Si la combinación de las diferentes opciones que el sistema provee en cada una de las preguntas que realiza al usuario, no es encontrada dentro de su base del conocimiento, entonces esto significa que el sistema necesita aprender esa nueva prótesis para que la próxima vez sea diagnosticada correctamente.

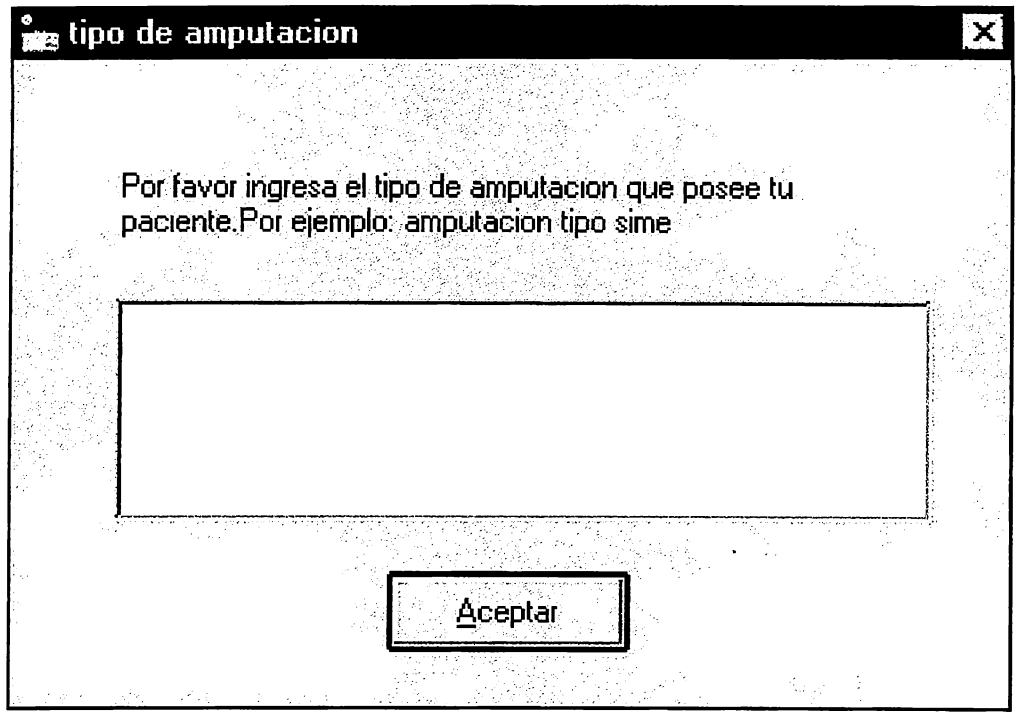
En caso que esta situación se de, el sistema presenta dos mensajes. En el primero le dice que no puede diagnosticar ese tipo de prótesis debido a que los datos son ingresados incorrectamente o no esta diseñado para diagnosticar ese tipo de prótesis, por tanto presenta un segundo mensaje en el cual le pide al usuario que se dispone a ingresar esta nueva prótesis a la base del conocimiento y que por favor sea lo más especifico posible al ingresar los datos. Estos mensajes son presentados respectivamente a continuacion:



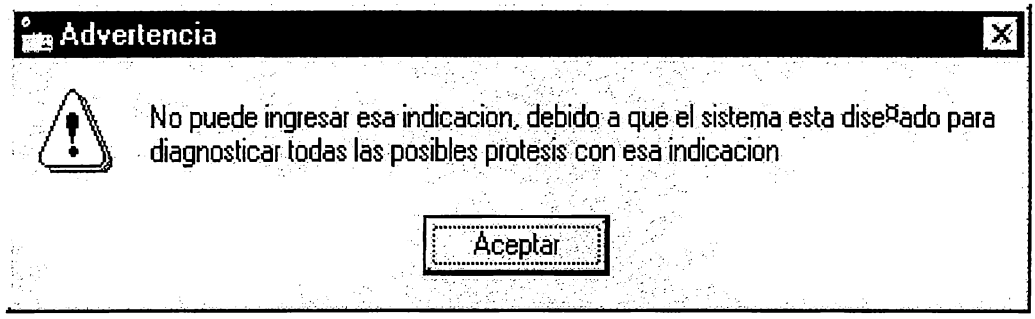


Una vez que se ha presionado Aceptar en este ultimo mensaje, el sistema muestra una nueva ventana llamada tipo de amputación en la cual se le pide que ingrese el tipo de amputación que un paciente

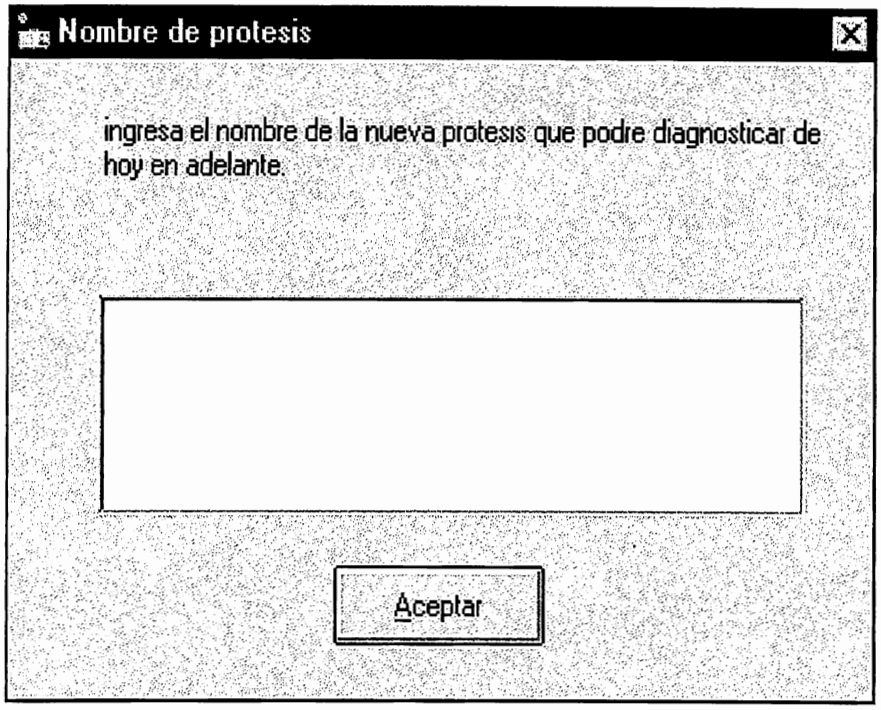
debe poseer para que se le diagnostique la nueva prótesis que se esta ingresando a la base del conocimiento:



En caso que el usuario ingrese un tipo de amputación que ya existe dentro de la base del conocimiento, el sistema rechaza esa petición y advierte al usuario que no puede ingresar ese tipo de amputación debido a que el sistema se encuentra diseñado para diagnosticar todas las posibles prótesis con ese tipo de amputación y por tanto el dato que se quiere ingresar es incorrecto. A continuación el sistema le pregunta nuevamente si desea realizar una nueva consulta con el sistema. El mensaje de advertencia que muestra el sistema se presenta a continuación:



En caso que el tipo de amputación sea valido, el sistema despliega una nueva ventana llamada nombre de prótesis, en dicha ventana se le pide al usuario que ingrese el nombre de la nueva prótesis que podrá ser diagnosticada con el tipo de amputación antes introducido.



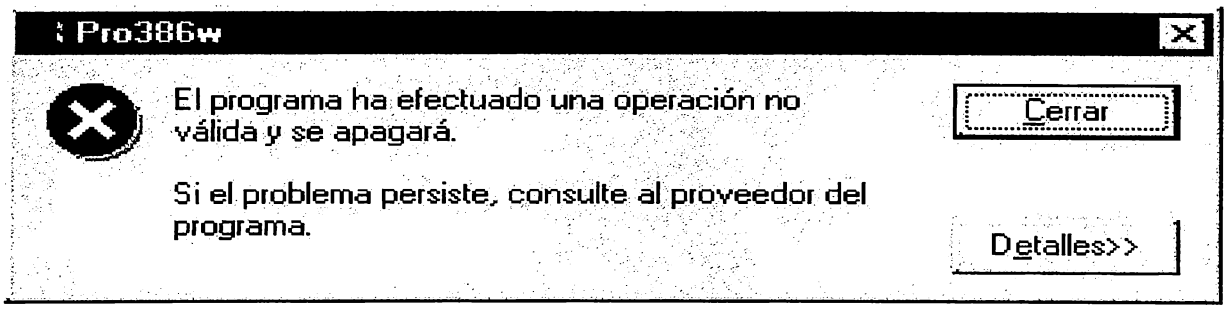
Una vez que se presiona aceptar, la nueva prótesis podrá ser diagnosticada posteriormente al presentar un paciente con el tipo de amputación introducido y finalmente el sistema le pregunta si desea realizar un nueva consulta.

SECCION V. RESOLUCION DE PROBLEMAS.

En las secciones anteriores se ha explicado la forma de cómo trabaja el Sistema Experto, los mensajes que aparecen en caso de realizar una acción incorrecta dentro del sistema y cuando y como se introducirá nueva información a la base del conocimiento.

En esta sección presentaremos el posible problema con el que se puede topar el ingeniero del conocimiento o programador en caso que el sistema falle.

El problema o error que puede aparecer es cuando el sistema operativo avisa que el programa ha ejecutado una acción invalida, tal y como se muestra en la siguiente figura:



La razón de que suceda este error es porque la memoria de la máquina que esta utilizando se ha llenado y ha producido un overflow por haber realizado una gran cantidad de diagnósticos y luego haber querido introducir una nueva prótesis a la base del conocimiento, por tanto debe reiniciar el equipo para que la memoria sea liberada y así estar disponible para realizar nuevas sesiones.

APENDICE D

MANUAL TECNICO DEL PROTOTIPO DE SISTEMA EXPERTO

Prototipo de Sistema Experto para el análisis y diagnóstico de Prótesis del Miembro Inferior

Versión 1.0

Manual Técnico

UNIVERSIDAD DON BOSCO

MARZO 1999

Introducción

El presente manual forma parte de la documentación dirigida al ingeniero del conocimiento encargado de la administración del Sistema Experto para el análisis y diagnostico de Prótesis del Miembro Inferior.

Requerimientos previos de la audiencia

Como condición previa a la lectura de este manual, se asume que el usuario tiene experiencia trabajando en ambientes Windows, ya sea con Windows NT o Windows 95 (todas marcas registradas de Microsoft Corp.). Además se asume que se tiene previamente instalado y posee los conocimientos de programación de Win-Prolog 2.6 versión del programador y flex shell toolkit (ambas marcas registradas por LPA). Por lo tanto, la funcionalidad del sistema asociadas con las características estándar del sistema operativo y el lenguaje de programación no son descritas.

¿ Cuál es el contenido de este manual?

El manual técnico se encuentra organizado en varias secciones:

Sección I, "Conocimientos Básicos sobre el Sistema", brinda una descripción de los puntos importantes que se deben conocer para empezar a entender la forma en que esta construido el prototipo de Sistema Experto para el análisis y diagnostico de prótesis del miembro inferior.

Sección II, "Estructura del Sistema", explica la estructura principal del sistema a través de la cual se podrá poner en funcionamiento el Prototipo de Sistema Experto para el análisis y diagnostico de prótesis del miembro inferior, además se explica cada una de las funciones que utiliza el sistema para trabajar correctamente y proporcionar las respuestas correctas.

SECCIÓN I. CONOCIMIENTOS BÁSICOS SOBRE EL SISTEMA

Para la construcción del prototipo del Sistema Experto para el análisis y diagnostico de prótesis del miembro inferior se hizo necesario utilizar el lenguaje de programación Win-Prolog y una de las herramientas de desarrollo o toolkit llamado flex expert system toolkit, el cual es una herramienta para Sistemas Expertos expresiva y poderosa que soporta razonamientos basados en frames con herencias, programación basada en reglas y procedimientos data-driven totalmente integrados dentro de un ambiente de programación lógica y contiene su propio Lenguaje KSL (English-like Knowledge Specification Language).

Flex emplea una arquitectura abierta y permite accesar, argumentar y modificar su comportamiento a través de una capa de funciones de acceso. Debido a esto es que flex es algunas veces conocido como un toolkit AI (herramienta de Inteligencia Artificial). La combinación de flex y prolog, por ejemplo una herramienta híbrida de Sistema Experto con un poderoso lenguaje de inteligencia artificial de propósito general , da como resultado un ambiente de desarrollo de sistemas expertos con una gran funcionalidad y versatilidad, donde los desarrolladores pueden afinar y mejorar el mecanimos de comportamiento construido internamente para cumplir sus propios requerimientos especificos.

El prototipo de Sistema Experto para el análisis y diagnostico de prótesis del miembro inferior es un Sistema Experto híbrido, ya que utiliza conjuntamente el lenguaje KSL de flex, como la sintaxis propia de win-prolog.

SECCION II. ESTRUCTURA DEL SISTEMA.

El prototipo de Sistema Experto esta constituido por dos archivos, uno con código del lenguaje ksl de flex toolkit y otro con código de win-prolog. Estos archivos se diferencian por las extensiones que utilizan, los de flex poseen ksl como extensión y los de win-prolog tienen pl como extensión, por tanto a partir de ahora nos referiremos a ellos como el archivo ksl y el archivo pl respectivamente.

El archivo ksl posee una función principal que es utilizada para poner en ejecución el sistema; Esta función principal se diferencia por poseer la palabra reservada action seguido por un nombre único que será a través del cual que el programa comenzará a ejecutarse. Un action es una colección de directivas o instrucciones a ejecutarse. Es similar a la definición de un predicado de prolog, excepto que solo una definición por cada acción es permitido. El código siguiente representa la estructura action:

```
action experto;
        do repeat
        and ask nombres
        and ask edad
        and indicación(Prótesis)
        and msgbox('respuesta',Protesis,48,Code)
        and adición(Prótesis)
        and close('sistemaexperto.pl')
        and compile('sistemaexperto.pl')
        and msgbox('Continuar','Desea Continuar?',36,Loop)
        until Loop is 7
```

end repeat.

El código del Sistema Experto esta constituido básicamente por 3 secciones, una sección de acciones, otra sección donde se encuentran las preguntas y respuestas del sistema y una última sección de relaciones que es donde se construye la base del conocimiento y esta bastante relacionada con la sección de las preguntas.

EL sistema realiza línea a línea el código escrito en el archivo ksl o pl, por tanto una vez que el sistema es llamado a través de la action experto la siguiente instrucción que realiza es la del loop repeat-until, es decir, que se repetirán las instrucciones subsiguientes, hasta que la condición que verifica al final del loop sea cumplida.

Tanto las preguntas, como las action y las relaciones pueden ser vistas como funciones, y pueden ser escrita en cualquier parte del archivo y en el orden que se desee, flex realiza una búsqueda a través de todo el archivo hasta que el nombre de la función concuerde con una que esta escrita en el código. Por lo tanto toda función en flex comienza ya sea con las palabras question, action o relation.

En esta sección se explicará detalladamente como se ejecuta el código del prototipo flex. Podría ser de bastante utilidad referirse al DFD del sistema que se encuentra en el capítulo de análisis y diseño del sistema para comprender exactamente que es lo que hace el sistema y como fluye la información dentro de él y sería bueno revisar el código completo del sistema que se encuentra completo en el apéndice B de este documento.

Como se menciona al principio de este manual, el sistema empieza a ejecutarse llamando a la action experto, luego invoca cada una de las directivas de la action de la siguiente manera: en primer lugar ingresa al loop repeat-until y continua invocando la question nombre y la question edad a través de los mandatos ask:

ask nombre y ask edad

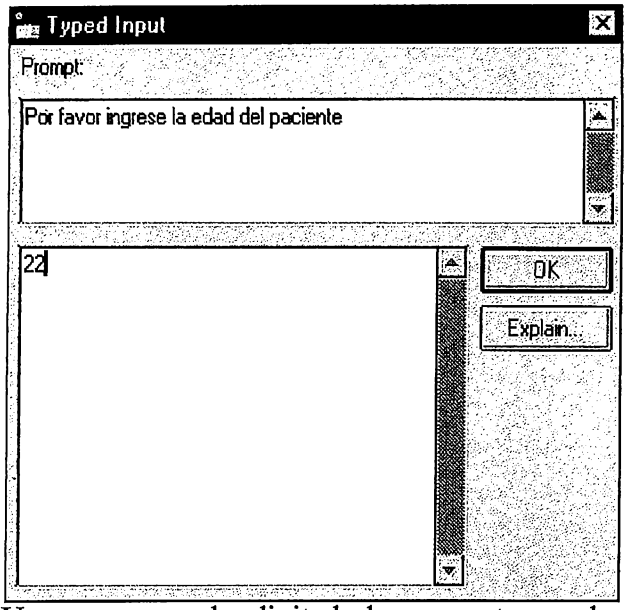
Una vez que ask invoca nombre, flex empieza a recorrer todo el código hasta que encuentra la función question nombre y question edad (recordemos que flex ejecuta primero una directiva y cuando se obtiene una respuesta, continua con la siguiente). El código que encuentra es el siguiente:

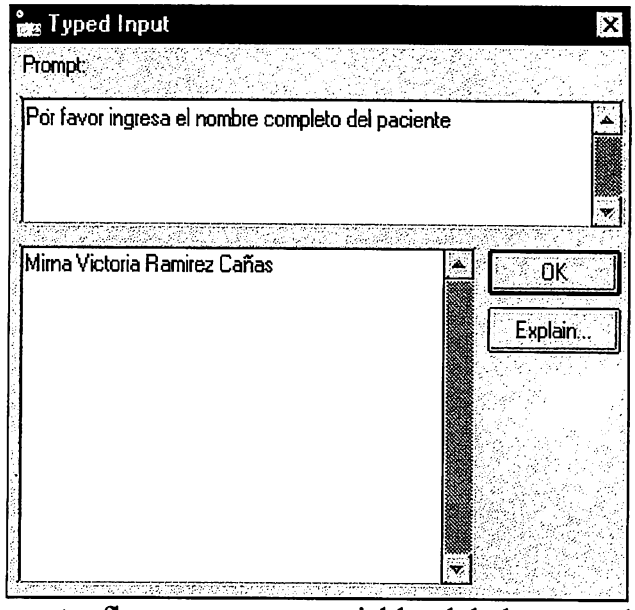
```
question nombres
        Por favor ingresa el nombre completo del paciente;
        input set
        because Es necesario llevar un registro de los datos personales del paciente.
question edad
        Por favor ingrese la edad del paciente;
        input integer
        because La edad es necesaria para diagnosticar algunas de las prótesis.
```

Estas preguntas son del tipo más simples que trae construidas internamente flex, se constituyen de la siguiente forma: en la primera línea se encuentra la palabra reservada question seguido del nombre con el cual son llamadas cada una, luego esta el texto que el sistema despliega como pregunta en la pantalla, en la siguiente línea va el tipo de dato que el sistema espera recibir como respuesta a su pregunta por parte del usuario; si la información que pretende ser introducida al sistema no es del tipo que este requiere en ningún momento se podrá ingresar los datos, por ejemplo en la primera pregunta se espera que la respuesta sea un set de caracteres, en cambio en la segunda forzadamente debe ser un entero el que deba ser ingresado.

Luego opcionalmente sigue la explicación a esa pregunta, es decir, el porque el sistema necesita esa información esto se logra precediendo a la explicación que se desee poner la palabra reservada because.

El código anterior genera automáticamente las siguientes pantallas:





Una vez que se ha digitado la respuesta a cada pregunta, flex genera una variable global para cada una con el mismo nombre que la función question y es guardada en memoria.

Una vez realizadas estas dos lineas de código, flex pasa a la siguiente línea donde llama a la función indicacion(Protesis), esto significa que debe encontrar en cualquier parte del código una función con el nombre indicacion y espera que regrese un valor que será asignado a la variable Protesis. En este caso la función que encuentra es una relation, las relations son definidas como una colección de clauses. Son relaciones logicas sobre entidades, similares a los predicados de prolog. La primera relation que encuentra es la siguiente:

relation indicación('prótesis tipo Sime de cuero')

if indica1 is 'amputación tipo sime'

and tipo is 'Estabilidad M-L'.

Como podrán darse cuenta, después de la palabra relation continua el nombre de la función que es llamada y lo que esta entre paréntesis es la data que será guardada en la variable Protesis, siempre y cuando se cumplan las clauses o condiciones necesarias para dar esa respuesta. Utilizando las

relations se ha construido la base del conocimiento de este prototipo, pudiendo haber sido mas factible crear esta base en una base de datos comercial y hacer las búsquedas hacia esta a través de ODBC desde prolog si se hubiera tenido otro toolkit de winprolog llamado datamite.

Luego que flex encuentra la primera relation, prosigue a probar si se cumplen el grupo de condiciones, en esta primera relación comienza buscando si la variable indica1 posee almacenado el valor "amputación tipo sime", inmediatamente flex se da cuenta que dicha variable no posee asignado valor alguno ya que en ningún momento ha sido inicializada antes y por tanto debe inferir en que parte del código se carga algún valor a indica1, recordemos que flex crea automáticamente una variable global con el mismo nombre que una question. Es de esta manera que es llamada la question indica1 presentada a continuación:

```
question indica1
        Cuál de las siguientes indicaciones posee el paciente?;
        choose one of grupo_indicaciones
        because Se necesita saber que indicación posee el paciente para
                determinar que prótesis necesita.
```

El programa en este momento ejecuta esta pregunta, presentado en primer lugar el texto de la pregunta en pantalla. La estructura de question indica1 es similar a la de question nombre y question edad, la diferencia es que las primerar esperaba un tipo especifico de respuesta por parte del usuario, en cambio en la estructura de question indica1 se sabe de antemano cuales son las posibles respuestas que el usuario ingresará al presentarle un menú de opciones a escoger. En este caso es un menú de selección simple, y este se crea al ejecutarse la linea "choose one of grupo_indicaciones", donde se le esta diciendo a flex que el usuario deberá escoger la respuesta de un grupo almacenado en grupo_indicaciones.

```
group grupo_indicaciones
```

'amputación tipo sime', 'amputación tipo pirogoff', 'amputación tipo Boyd', 'amputación trans tibial', 'desarticulación de Rodilla', 'amputación trans femoral', 'desarticulación de cadera', "hemipelvectomia'.

Un group no es más que una colección de datos que pueden ser almacenados juntos. Una vez se ha presentado en pantalla las distintas opciones, el usuario podrá escoger una de ellas en este caso, haciendo click sobre dichas opciones.

La respuesta seleccionada es almacenada en indica1 por las razones explicadas anteriormente, y flex retorna a la relation que estaba probando.

Es en este momento cuando verifica, si la respuesta guardada en indica1 es el valor con el cual se le esta comparando, en este caso "amputación tipo sime", si es asi prosigue a probar la segunda condición de esa misma relación hasta que todas sean ciertas para poder enviar la respuesta; en el caso que la primera de ellas o cualquier otra resulte falsa, flex inmediatamente deja de seguir probando esa relación y busca otra cuyas condiciones (cualesquiera que sean) resulten verdaderas.

Al almacenar la respuesta en la variable Protesis, flex continua con la siguiente instrucción, que en este caso es la de desplegar en pantalla una caja de mensaje conteniendo el diagnostico encontrado con el análisis previo de las respuestas a las preguntas que se hicieron y luego se procede a preguntar si se desea realizar una nueva sesion o no, para probar la condición del loop repeat-until.

El párrafo anterior explica exactamente como trabaja flex con las questions, relations y las actions, en el caso que el sistema desplegará un respuesta valida, ahora veamos como trabaja el sistema cuando las respuestas dadas por el usuario no corresponden a ningún tipo de prótesis existente dentro de su base del conocimiento. Es aquí donde se añade nueva información a la base del conocimiento haciendo uso de un archivo pl y por lo tanto el sistema empieza a aprender del usuario.

relation indicación('El sistema no puede diagnosticar una prótesis con las indicaciones dadas, debido a: no esta diseñado para diagnosticar ese tipo de prótesis o las indicaciones no son las correctas para una prótesis existente')

if indica1 is 'amputación trans tibial'

and nivel is '1/3 distal'

and [indica2 is '4) Existe ruptura de ligamentos en la rodilla del miembro amputado'].

Como se explico anteriormente, si todas las condiciones de una relation son verdaderas, esta prosigue a almacenar la data que esta entre paréntesis en la variable Protesis. Al almacenar la información de la relation que se presenta al principio de este párrafo, flex retorna su ejecución al punto donde fue llamada *"indicacion(Protesis)"* y prosigue con la siguiente línea que es presenta la información almacenada en Protesis a través de una cuadro de mensaje y luego llama a *"adicion(Protesis)"*. Con esta función se pretende añadir a la base del conocimiento el nombre de la prótesis que hace falta, junto con su característica o indicación principal. Por lo tanto, flex realiza nuevamente la búsqueda de la función anterior de igual forma como se explico en la parte de las relation, la única diferencia es que en este caso no encontrará una relation, sino una action, la cual es presentada a continuación:

action adición(Prótesis);

do if Prótesis is 'El sistema no puede diagnosticar una prótesis con las indicaciones dadas, debido a: no esta dise¤ado para diagnosticar ese tipo de protesis o las indicaciones no son las correctas para una protesis existente'

then ensure_loaded(sistemaexperto)

and prolog(_)

else msgbox('Experto','Diagnostico finalizado',48,Si)

end if.

Al igual que la action principal del sistema, esta contiene directivas que deben ser probadas una a una, en este caso, si la condición de que la información contenida en la variable Protesis es la información con la que se compara, flex carga a memoria el archivo pl llamada sistemaexperto.pl, luego llama el predicado prolog(_) de ese archivo.

Es en este momento que el sistema experto se convierte en uno híbrido dado que combina el lenguaje ksl de flex con el lenguaje pl de winprolog. Dado que flex ha sido construido con el lenguaje prolog, por tanto el comportamiento en cuanto a la ejecución es la misma, lo único que cambia es la sintaxis que se utiliza.

Tal y como se presento al inicio de esta sección la estructura o función principal del sistema, ahora presentamos el predicado principal del archivo pl:

```
prolog(_):-
msgbox('Creación de nueva prótesis: ', 'A continuacion se procedera a ingresar a la base de conocimietos la nueva informacion. Favor responder las preguntas lo mas especifico posible',48,Code),
        indicación,
        close('sistemaexperto.ksl'),
        compile('sistemaexperto.ksl').
```

La primera instrucción importante que se ejecuta desde este archivo es la llamada al predicado indicacion (notar que dado que este es un archivo pl, se utilizarán términos propios de prolog), el cual a su vez realiza una llamada al predicado ventana1:

```
ventana1:-
        _S1 = [dlg_ownedbydesktop,ws_sysmenu,ws_caption],
        _S2 = [ws_child,ws_visible,ss_left],
```

_S3=[ws_child,ws_visible,ws_tabstop,ws_border,es_left,es_multiline,es_autohscroll,es_auto vscroll],

_S4 = [ws_child,ws_visible,ws_tabstop,bs_pushbutton],

wdcreate(amputación, `tipo de amputacion`,235, 43, 372, 261, _S1),

wccreate((amputacion,1000), static, `Por favor ingresa el tipo de amputación que posee tu paciente. Por ejemplo: amputación tipo sime`,40, 40, 310,40, _S2),

wccreate((amputacion,800), edit,``,40, 90, 300, 80, _S3),

wccreate((amputacion,1),button, `&Aceptar`, 140, 190, 90, 30, _S4),

window_handler(amputacion,amputacion_handler),

call_dialog(amputacion,Result),

ensure_loaded('sistemaexperto.ksl'),

intento(Result).

La ventana que es construida con este predicado, es utilizada para que el usuario ingrese el nombre de la indicación que debe añadirse al grupo_indicaciones para que aparezca entre las opciones a escoger cuando se realice la pregunta la próxima vez que se ejecute el programa. Una vez que el usuario a digitado la información y presionado el botón aceptar, esta es almacenada en una variable llamada Result a través del manejador de ventana siguiente:

amputacion_handler((amputacion,1),msg_button,_,Result):-

wtext((amputacion,800),Res),

atom_string(Result,Res).

Luego es cargado a memoria nuevamente el archivo ksl y es llamada la action intento(Result), donde es pasado por parametro por valor la variable Result que corresponde a la respuesta nueva del usuario.

Esta relation lo que hace es verificar si esta nueva respuesta del usuario efectivamente no pertenece a la base del conocimiento, ya que el sistema posee toda la información existente hasta ese momento sobre prótesis y no puede aceptar información repetida. La forma de cómo ejecuta el siguiente trozo de código es similar a como se ha explicado hasta ahora. Lo que sucede en esta action es que se asigna a la variable "a" el contenido de la variable Result, luego a través del comando catch es almacenado en la variable V el valor que resulta de ejecutar la relation int(Res) que es donde se verifica si "a" esta repetido o no, es decir que si las condiciones de la relación resultan verdaderas esta arroja un valor de 0 que es almacenado en V, caso contrario se almacena el valor de –1. Después se llama a la relation relación(V,Res,Result), donde son pasado a través de parámetros por valor la información contenida en V, Res y Result. Aquí es donde se manda el mensaje que esa información no puede ser ingresada porque ya existe y sino el cargado nuevamente a memoria el archivo pl y ejecutado el predicado ventana2.

```
action intento(Result);
do a become Result
and catch(V,int(Res))
and relación(V,Res,Result).
action relación(V,Res,Result);
do if V is 0
then msgbox('Advertencia',Res,48,Codigo)
else ensure_loaded(sistemaexperto)
     and ventana2(Result)
end if.
```

relation *int('No puede ingresar esa indicación, debido a que el sistema esta diseñado para diagnosticar todas las posibles prótesis con esa indicación')*

if [a is 'amputación tipo sime' or a is 'amputación tipo pirogoff' or a is 'amputación tipo Boyd' or a is 'amputación trans tibial' or a is 'desarticulación de Rodilla' or a is 'amputación trans femoral' or a is 'desarticulación de cadera' or a is 'hemipelvectomia'].

El predicado ventana2 ejecuta una segunda ventana que es para que el usuario ingrese el nombre de la nueva prótesis y así poder empezar a adicionar toda la información recopilada al archivo ksl que equivale a la base del conocimiento.

Una vez que se ha almacenado la información ingresada en esta ventana en una variable a través del manejador de ventana respectivo, procede a calcular la longitud de esta, con el propósito de ir sabiendo en que nueva posición debemos escribir cuando sea requerido. Para esto se ha utilizado dos archivos de texto (puntero y puntero1), desde donde se lee la nueva posición donde se debe ubicar el puntero para escribir en el archivo ksl y donde se debe escribir la última posición donde termino el puntero para que sirva como nueva posición en la siguiente sesión. En el archivo puntero1 se ha almacenado la posición del puntero donde se añadirá la información de verificación de datos existentes, y en el archivo puntero se encuentra la nueva posición donde debe ubicarse el puntero para añadir la nueva indicación al grupo_indicaciones.

ventana2(Result):-

_S1 = [dlg_ownedbydesktop,ws_sysmenu,ws_caption],

_S2=[ws_child,ws_visible,ws_tabstop,ws_border,es_left,es_multiline,es_autovscroll,es_auto hscroll],

_S3 = [ws_child,ws_visible,ws_tabstop,bs_pushbutton],

_S4 = [ws_child,ws_visible,ss_left],

wdcreate(relation, `Nombre de protesis`,235,43,372,261,_S1),

wccreate((relation,800),edit,``,40,90,300,80,_S2),

wccreate((relation,100),button,`&Aceptar`,140,190,90,30,_S3),

wccreate((relation,1000),static, `ingresa el nombre de la nueva prótesis que podré diagnosticar de hoy en adelante.`,40,20,310,40,_S4),

window_handler(relation,relation_handler), % inicialización del manejador de ventana

call_dialog(relation,Relation), % llamada al manejador de ventana

write('~M~J')~>'sistemaexperto.ksl', % todos las instrucciones

write('relation indicación("")~>'sistemaexperto.ksl', % write a las que le siguen

write(Relation)~>'sistemaexperto.ksl', % el simbolo '~>' están

write('")~M~J')~>'sistemaexperto.ksl', % escribiendo la información

write('if indica1 is ")~>'sistemaexperto.ksl', % contenida entre paréntesis

write(Result)~>'sistemaexperto.ksl', % al final del archivo ksl

write('".')~>'sistemaexperto.ksl',

write('~M~J')~>'sistemaexperto.ksl',

write('~M~J')~>'sistemaexperto.ksl',

len(Result,L), % instrucción para determinar la longitud de la variableResult

see('puntero'), % abriendo el archivo de texto puntero para leer su información

fread(i,0,0,I), % comando para leer en formato integer y almacenar ese dato en I

seen, % instrucción para cerrar el archivo abierto anteriormente para lectura

A is I+L+1,

tell('sistemaexperto.ksl'), %instrucción para abrir el archivo para escribir en el

outpos(I), %comando para posicionar el puntero en la posición I

write(',"), %En este caso se utiliza write para escribir en el archivo abierto a partir de I

write(Result),

write('".'),

told, %Instrucción utilizada para cerrar el archivo abierto anteriormente para escritura

tell('puntero'),

outpos(0),

fwrite(i,0,0,A),

told,

close('puntero'),

see('puntero1'),

fread(i,0,0,J),

seen,

tell('sistemaexperto.ksl'),

B is L+J,

outpos(J),

write(' or a is "'),

write(Result),

write('"].'),

told,

tell('puntero1'),

outpos(0),

fwrite(i,0,0,B),

told,

close('puntero1').

Una vez que se ha añadido la información en el archivo ksl, se regresa el mandato a la función principal del sistema en el archivo ksl al seguir ejecutando el predicado prolog(_) desde donde se había quedado y cargando a memoria nuevamente al archivo ksl. Al regresar al archivo ksl se continúan ejecutando las siguientes instrucciones desde donde se había quedado hasta completar el loop repeat-until, tal y como se explico anteriormente en este manual.

GLOSARIO

A continuación se presentan algunos conceptos básicos en el área de la Inteligencia Artificial:

- **BASE DEL CONOCIMIENTO.**

  De un Sistema Experto contiene el conocimiento de los hechos y de las experiencias de los expertos en un dominio determinado.

- **BACKWARD CHAINING.**

  Encadenamiento hacia atrás, se plantean hipótesis y se intentan demostrar con información conocida, es decir se llega a una respuestas a partir del objetivo que se busca.

- **CALCULO DE PREDICADOS.**

  Deducción lógica de resultados, mediante el cumplimiento de determinadas condiciones puede extraerse una deducción lógica. La solución puede tener el valor verdadero o falso.

- **COMPONENTE EXPLICATIVO.**

  Explica al usuario la estrategia de solución encontrada y el porque de las decisiones tomadas.

- **ENTREVISTA.**

  Es la comunicación interpersonal establecida entre el investigador y el sujeto de estudio, con el fin de obtener respuestas verbales a las interrogantes sobre el tema propuesto. Esta técnica permite captar información abundante y básica sobre el problema.

- **EPISTEMOLOGICO.**

  Doctrina de los fundamentos y métodos del conocimiento científico.

- **FACTORES DE CERTEZA.**

  Son factores arbitrarios de valoración que suelen encontrarse casi siempre dentro de los limites de –1 a +1.

- **FRAMES.**

  Estructuras de datos para la representación de objetos.

- **HEURISTICA.**

  Arte de inventar.

- **INTELIGENCIA ARTIFICIAL.**

  Define las técnicas de la lógica formal, de los nuevos procedimientos y métodos de búsqueda de la base de la representación del conocimiento en programas de ordenador, por otro lado se define la Inteligencia Artificial por la presencia aparente de experiencia y conocimiento de causa en programas de ordenador.

- **INTERFASE.**

  Es la que mantiene el diálogo de pregunta y respuesta entre el sistema y el usuario.

- **KSL.**

  English-like Knowledge Specification Language, este lenguaje es utilizado para definir reglas, frames y procedimientos dentro de flex. Es un lenguaje bastante expresivo, fácil de programar y fácil de leer. El KSL permite al programador escribir oraciones simples y concisas acerca del mundo del experto que luego pueden ser entendidas y mantenidas por personas que no son programadores.

- **LENGUAJES DE PROGRAMACION SIMBOLICAS.**

  Son especialmente apropiados para la representación formalizada del conocimiento. Son interpretativos y orientados al diálogo.

- **LISP.**

  (List-Processing) es un lenguaje funcional, ofrece la posibilidad de realizar definiciones recursivas funciones. Lenguaje de programación que procesa listas; utilizando, entre otras

aplicaciones, para el desarrollo de sistemas Expertos. La principal característica es su capacidad de manejar símbolos y estructuras.

- **PROGRAMACION ORIENTADA A OBJETOS.**

  Es el proceso de programación con el que puede describirse la información como objetos. Los objetos son los determinados por sus cualidades y su comportamiento.

- **PROLOG.**

  Es una abreviatura de PROgrammimg in LOGic, se aplica como lenguaje de desarrollo en aplicaciones de la Inteligencia Artificial en diferentes proyectos, se limita a las partes necesarias.

- **REGLAS DE PRODUCCION.**

  Se basan en la lógica de predicados.

- **REDES SEMANTICAS.**

  Es una representación gráfica del saber sobre los objetos y sus relaciones.

- **SHELL.**

  Son herramientas que facilitan el trabajo del ingeniero del conocimiento.

- **SISTEMA EXPERTO.**

  Es una expresión basada en el conocimiento, el término de Sistemas Expertos se entiende como un nuevo tipo de Software que imita el comportamiento de un experto humano en la solución de un problema. Pueden almacenar conocimientos de expertos para un campo determinado y muy limitado, solucionar un problema mediante deducción lógica de conclusiones.

- **VISUAL BASIC.**

  Proporciona un ambiente de desarrollo donde se trabaja con objetos y eventos, llega a ser un proceso directo y, lo más importante, bien estructurado. Es utilizado para la creación y diseño de

bases de datos relacionales con la utilización de objetos de datos los cuales son usados para manipulas estas bases, las tablas e índices.

- **VISUAL FOX.**

  Es un sistema de administración de base de datos (DBMS); su propósito es ayudar a reunir, recuperar (o extraer) y presentar datos. Una de las ventajas de utilizar sistemas de administración de bases de datos por computadora son: 1) pueden almacenar grandes cantidades de datos y 2) que permiten extraer y reorganizar rápidamente los datos.

- **WIN-PROLOG.**

  Es un compilador de prolog de 32-bit y un ambiente de programación.

BIBLIOGRAFIA

- Peter S. Sell, "Sistemas Expertos para Principiantes", 1992, Editorial Limusa, S. A. de C.V .

- Dieter Neebendahl, "Sistemas Expertos", 1988, Marcombo Boixareu Editores.

- Guillermo Ernesto Cortes Vileda, "Desarrollo de un Sistema Experto para Orientación Vocacional", 1992.

- Dheeraj (Raj) Khera, "The Impact Of Expert System",1992

- Julie Wallin Kaewert, "Developing Expert System For Manufacturing", 1990, McGraw-Hill

- Jerome T. Murray, "Expert Systems in data processing", 1998, McGraw Hill.

- George F. Luger, "Artificial inteligence and the design of Expert Systems", 1989, Benjamin Cummings.

- Carl Townsenc, "Introduction to Turbo Prolog", SyBex.

- Daniel Cohen, "Sistems de Información para la toma de decisiones", 1994, McGraw Hill

- Jesús Cardeñosa Lera, "La ingeniería del conocimiento en el umbral del siglo XXI", 1990, Universidad Politécnica de Madrid.
- Jesús Cardeñosa Lera, "Adquisición y Representación del Conocimiento", 1990, Universidad Politécnica de Madrid.
- Roger S. Pressman, "Ingeniería del Software", 1998, McGraw-Hill/Interamericana de España, S.A.U.
- R. Viladot, O. Cohi, S. Clavell, "Ortesis y Prótesis del Aparato Locomotor", 1997, Masson, S.A.
- Dave Westwood, "LPA-PROLOG programming guide", 1994.
- Brian D. Steel, "Win32 Programming guide", 1992-1997.
- Dave Westwood, "Win-Prolog 3.5 Techincal Reference", 1997.
- Nicky Johns, "Flex Expert System Toolkit", Enero de 1997