



UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

SEDE VICTOR LEVI SASSO



HERRAMIENTAS APLICADAS A LA INTELIGENCIA ARTIFICIAL

INCLUYE PRUEBAS SUMATIVAS Y PRESENTACIONES DEL CONTENIDO

DR. CARLOS A. ROVETTO

MAYO 2018



Universidad Tecnológica de Panamá (UTP)

Esta obra está licenciada bajo la Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Para ver esta licencia:

<https://creativecommons.org/licenses/by-nc-sa/4.0>

Contenido

Índice de figuras	5
Introducción.....	8
Capítulo I: Herramientas para crear y ejecutar sistemas inteligentes.....	10
Herramientas y técnicas para el desarrollo de sistemas inteligentes.....	11
Ventajas de los Shells y los productos de Sistemas Inteligentes	13
Alternativas de desarrollo de Sistemas Inteligentes	14
Aplicaciones de los Sistemas Inteligentes y la IA	15
Guía evaluativa de herramientas para crear Sistemas Inteligentes.....	19
Estructura de la evaluación	19
Características de la aplicación.....	19
Capacidades de la herramienta	20
Métricas	20
Técnicas de evaluación.....	21
Contextos.....	22
Metodología	22
Pasos para evaluar herramientas para crear Sistemas Inteligentes.....	23
Paso 1: Determinar las características de la aplicación	23
Paso 2: Identificar los contextos relevantes	23
Paso 3: Derivar las capacidades significativas de la herramienta.....	23
Paso 4: Identificar las métricas discriminantes y las técnicas de evaluación	23
Paso 5: Identificar las herramientas disponibles	23
Paso 6: Filtrar las herramientas disponibles para identificar a las herramientas candidatas.....	23
Paso 7: Podar y priorizar cada una de las dimensiones	23
Paso 8: Aplicar el esquema de la estructura para evaluar y seleccionar herramientas	24
Capítulo II: Desarrollo de aplicaciones con Expert System Builder	25
¿Qué es Expert System Builder?.....	25
Etapas de desarrollo del Expert System Builder.....	27
Capítulo III: Desarrollo de aplicaciones con Exsys Corvid	29
Áreas de aplicación de la herramienta.....	31

Construcción de una aplicación con Exsys Corvid	32
Primer paso: Elección de un problema y descomposición en pasos lógicos	33
Pantalla de variables.....	34
Definiendo valores de las variables Corvid	38
Definición y representación gráfica	39
Ventana de comandos	42
Segundo paso: Añadir bloques lógicos.....	43
Bloques de control	46
Tercer paso: Añadir bloques de comandos	49
Cuarta etapa: Ejecución	52
Capítulo IV: Desarrollo de aplicaciones con Visual Prolog	54
Características del lenguaje	55
Fundamentos de Prolog	56
PROgramando en LOGica	56
Del lenguaje natural a los programas Prolog	60
Programas Visual Prolog	69
Entorno de desarrollo Prolog	70
Secciones básicas de un programa Visual Prolog	71
Unificación y backtracking.....	78
Capítulo V: Desarrollo de aplicaciones con CLIPS	84
Introducción a CLIPS.....	84
¿Qué es CLIPS?	84
Elementos básicos de una herramienta de un Sistema Inteligente.....	87
Entrada y salida de CLIPS	89
Elementos básicos de CLIPS	92
Hechos en CLIPS.....	93
Reglas en CLIPS.....	95
Variables en CLIPS.....	98
La cola de activaciones.....	100
Hechos más complejos	100
Otros aspectos de control	103
Bibliografía	109

Anexos 1: Pruebas Rápidas.....	112
Anexos 2: Presentaciones.....	123

Índice de figuras

Figura 1. Estructura de un sistema experto. - Tomado de (Badaró, Ibáñez, & Agüero, 2013).	18
Figura 2. Interfaz de ES-Builder.	25
Figura 3. Ejemplo del árbol de decisión de un sistema experto en ES-Builder.	28
Figura 4. Acceso a la pantalla de variables a través del ícono "Variables".	34
Figura 5. Acceso a la pantalla de variables a través de "Windows".	35
Figura 6. Pantalla de variables.	35
Figura 7. Área de formato de variables.	36
Figura 8. Área de lista de variables.	37
Figura 9. Área de valores de las variables.	37
Figura 10. Selección del botón "New".	38
Figura 11. Ventana "New Variable".	38
Figura 12. Ejemplo de un sistema donde se utiliza una variable de confianza. - Tomado de (Exsys Corvid, 2007).	40
Figura 13. Área de lista de variables donde se muestran las variables de confianza "Coast_Road" y "City_Streets". - Tomado de (Exsys Corvid, 2007).	41
Figura 14. Diagrama de árbol de la toma de decisiones donde se incluyen dos de las variables de confianza añadidas al sistema. - Tomado de (Exsys Corvid, 2007).	41
Figura 15. Bloque lógico que representa en Corvid las reglas IF-THEN del sistema. - Tomado de (Exsys Corvid, 2007).	42
Figura 16. Ventana de comandos "Display Commands". - Tomado de (Exsys Corvid, 2007).	43
Figura 17. Botón para acceder a la ventana de bloques lógicos. – Tomado de (Exsys Corvid, 2011).	44
Figura 18. Ventana "Logic Block".	44
Figura 19. Nombre del bloque lógico.	44
Figura 20. Área de trabajo y botones de cortar, copiar y pegar en la parte superior.	45
Figura 21. Área de nodo.	45
Figura 22. Control "MetaBlock".	45
Figura 23. Botón "Find".	45

Figura 24. Ventana "Rule View".	46
Figura 25. Controles para agregar nodos IF-THEN.	46
Figura 26. Pantalla "Add To Block".	47
Figura 27. Ejemplo del formato de un nodo en el bloque lógico. – Tomado de (Exsys Corvid, 2011).	47
Figura 28. Ejemplo de condiciones IF. – Tomado de (Exsys Corvid, 2011).	48
Figura 29. Ejemplo de condiciones IF con condiciones dentro de ellas. – Tomado de (Exsys Corvid, 2011).	48
Figura 30. Ejemplo donde se muestra un nodo THEN. – Tomado de (Exsys Corvid, 2011).	48
Figura 31. Botón para acceder a la ventana "Command Block". – Tomado de (Exsys Corvid, 2011).	49
Figura 32. Ventana "Command Block".	50
Figura 33. Botones "Control".	50
Figura 34. Botones "Command".	50
Figura 35. Ventana "Commands".	51
Figura 36. Ejemplo de un bloque de comando simple.	52
Figura 37. Ventana "Build Command File".	52
Figura 38. Botón "Run". – Tomado de (Exsys Corvid, 2011).	52
Figura 39. Ejemplo de plantilla predeterminada para la página web de un sistema creado en Corvid.	53
Figura 40. Ejemplo de sintaxis de un hecho en Prolog.	57
Figura 41. Ejemplo de sintaxis de una regla en Prolog.	58
Figura 42. Ejemplo de la sintaxis de una regla con dos objetivos.	58
Figura 43. Ejemplo de la sintaxis de una consulta en Prolog.	59
Figura 44. Ejemplo de una consulta utilizando una variable.	59
Figura 45. Ejemplo de una respuesta dada por Prolog.	60
Figura 46. Ejemplo de más de una respuesta mostrada por Prolog.	60
Figura 47. Diagrama que será utilizado como ejemplo para llevar de lenguaje natural a Prolog.	61
Figura 48. Ejemplo de un comentario de una sola línea.	69

Figura 49. Ejemplo de un comentario de más de una línea.	69
Figura 50. Entorno de desarrollo de Visual Prolog. - Tomado de Visual Prolog-Wikipedia.	71
Figura 51. Funcionamiento de CLIPS con el algoritmo RETE. - Adaptado de (Crowley, 2017).	88
Figura 52. Componentes de un sistema experto desarrollado en CLIPS.	88
Figura 53. Ícono de CLIPS en Windows.	90
Figura 54. Ventana con cuadro de diálogo "Dialog Window" en CLIPS. – Tomado de (Cubero & Berzal, 2011).	90
Figura 55. Ejemplo del uso de comandos que se pueden ingresar en la Ventana de Diálogo.	91
Figura 56. Menú en la parte superior de la Ventana de Diálogo desde el cual pueden utilizarse algunos comandos. – Tomado de ("CLIPS Tutorial 1," 2009).	92

Introducción

La Inteligencia Artificial persigue como objetivo principal reproducir o integrar en una máquina la capacidad de raciocinio nata de los seres humanos, capaces de discernir entre una serie de opciones la que mejor se adapte a la situación que tiene en frente; mediante la aplicación de una enorme gama de algoritmos que se han ido mejorando con el pasar de las décadas.

El campo de la Inteligencia Artificial capta la atención de muchos con solo escuchar su nombre, a raíz de esta fascinación, ingenieros, científicos y otra gama de profesionales trabajan constantemente para generar nuevo conocimiento dentro de esta rama de la ciencia. Para esto es necesario contar con herramientas de programación, sobre todo para aplicar los algoritmos que surjan a medida que avancen los estudios realizados, no obstante, los lenguajes de programación comunes, a pesar de contar con prestaciones para esta labor, requieren de un mayor tiempo, conocimiento y experiencia por parte de los desarrolladores para completar los proyectos, motivo que da marcha a la tangible necesidad de crear herramientas de software dedicadas a la creación de sistemas expertos.

Como era de esperarse las primeras versiones de herramientas destinadas a la programación de aplicaciones con inteligencia artificial trabajaban mediante consolas de comandos, lo que hacía algo complicado su uso, además de ser creados y para uso exclusivo en proyectos de investigación en laboratorios cerrados al público en general. Posterior a esto, con los avances en la informática general, se da paso a la creación de herramientas amigables al usuario, accesibles a todo público, colocando a disposición de cualquier persona la documentación para utilizarlas, así como su descarga desde Internet.

En el presente folleto se presentan las herramientas más comunes para la creación de sistemas expertos, unas gratuitas y otros que ofrecen periodos de prueba para su posterior adquisición tras comprar una licencia.

Se describen los aspectos generales de cada herramienta, entre los que se contemplan elementos históricos del surgimiento y su desarrollo, tipos de datos, estructuras de control y ejemplos de sintaxis de cada entorno. Por otra parte, en el caso de los softwares con una interfaz de usuario, se describen los bloques o secciones que lo componen, resaltando su papel dentro del proceso de creación de un sistema experto a través de estas.

Se ofrecen ejemplos ilustrativos del uso de cada herramienta haciendo énfasis en el orden en que se deben realizar las declaraciones y procesos dentro del sistema, mediante el uso de imágenes tomadas en algunos casos de manuales proporcionados por los desarrolladores o compañías a los que pertenece el software.

Todo esto con el propósito de mantener una guía lo más explícita para orientar a los estudiantes a través de sus primeras experiencias en la creación de sistemas expertos.

Capítulo I: Herramientas para crear y ejecutar sistemas inteligentes

Para comprender el concepto de sistema inteligente se debe empezar por definir dos conceptos: sistema e inteligencia. Un sistema es un conjunto de elementos organizados que se relacionan entre sí para lograr un objetivo, mientras que inteligencia se entiende como la capacidad de comprender y de resolver problemas utilizando el razonamiento.

En este sentido, un sistema es inteligente cuando tiene la capacidad de interpretar información, de comprender las relaciones entre objetos y fenómenos, ejecutar procedimientos significativos y aplicar información adquirida a partir de diferentes condiciones (Dahiya, Chaudhary, Saini, & Kumar, 2015). Aparte debe contar con otros requisitos tales como seguridad, conectividad, robustez y proveer respuestas en base a las circunstancias de su entorno.

Con el desarrollo de la Inteligencia Artificial (IA) y la evolución tecnológica, los sistemas inteligentes deben ser capaces de responder ante las exigencias de los usuarios a través de inferencias. A partir de esto surge la necesidad de incorporar en el desarrollo de estos sistemas herramientas y técnicas que permitan resolver problemas tales como la interoperabilidad, confiabilidad y la calidad del servicio.

El paradigma de la computación convencional se queda corto, ya que para que un sistema sea considerado inteligente se debe ir más allá de un algoritmo que le indique al computador cómo resolver un problema. Se debe utilizar el paradigma de la computación en IA que implica brindarle conocimiento y la capacidad de inferir al computador y un programa capaz de determinar el procedimiento específico para generar una solución.

La siguiente tabla permite visualizar una comparación entre un sistema clásico y un sistema experto, que es la base de los sistemas inteligentes y es lo que se pretende lograr con el paradigma de la computación en IA.

Sistema clásico	Sistema experto
Conocimiento y procesamiento combinados en un programa.	Base de conocimiento separada del mecanismo de procesamiento.

No contiene errores	Puede contener errores
No da explicaciones, los datos sólo se usan o escriben	Una parte del sistema está formado por el módulo de explicación
Hacer cambios es tedioso	Los cambios en las reglas son fáciles de hacer
El sistema sólo opera completo	El sistema puede funcionar con pocas reglas
Se ejecuta paso a paso	La ejecución usa heurísticas y lógica
Necesita información completa para operar	Puede operar con información incompleta
Representa y usa datos	Representa y usa conocimiento

Herramientas y técnicas para el desarrollo de sistemas inteligentes

En algunos casos el sistema experto es uno de los módulos con un rol específico y significativo en todo sistema inteligente en general; en otros casos, el sistema experto puede ser el componente principal del que dependen todos los demás. Independientemente del rol del sistema experto, el desarrollador debe estar consciente de cómo procederá en el desarrollo de este componente (Bielawski & Lewand, 1991).

Las herramientas y técnicas para el desarrollo de sistemas inteligentes son variadas y la integración de ellas permite lograr sistemas de alto nivel. En general, el proceso de desarrollo de sistemas expertos se realiza en función de la naturaleza del sistema que se construye, la estrategia de desarrollo y las herramientas de apoyo.

El ciclo de vida de un sistema experto se compone de seis fases que constituyen procesos no lineales para el desarrollo del sistema:

Fases para el desarrollo del sistema		
1	Inicialización del proyecto	<ul style="list-style-type: none"> • Definición y evaluación del problema • Evaluación de alternativas de soluciones • Verificación de un enfoque de sistemas expertos

		<ul style="list-style-type: none"> • Estudio de factibilidad • Análisis coste beneficio • Examen de las cuestiones de gestión • Organización del equipo de desarrollo
2	Análisis y diseño de sistemas	<ul style="list-style-type: none"> • Diseño conceptual y plan de trabajo • Estrategia de desarrollo • Fuentes de conocimiento • Recursos informáticos
3	Prototipado rápido	<ul style="list-style-type: none"> • Construcción de un pequeño prototipo • Pruebas, mejoras y ampliación • Demostración y análisis de viabilidad • Diseño completo
4	Desarrollo del sistema	<ul style="list-style-type: none"> • Desarrollo de la base de conocimientos • Definición de posibles soluciones • Definición de los hechos de entrada • Desarrollo de un esquema • Dibujo del árbol de decisión para representar visualmente el conocimiento • Creación de un mapa de conocimiento • Prueba, evaluación y mejoras de la base de conocimiento • Creación del plan de integración
5	Implementación	<ul style="list-style-type: none"> • Aceptación por los usuarios • Instalación, demostración y despliegue • Entrenamiento de orientación • Seguridad • Documentación • Integración y pruebas de campo
6	Post-implementación	<ul style="list-style-type: none"> • Operaciones • Expansión: Mantenimiento y actualizaciones

- | | | |
|--|--|---|
| | | <ul style="list-style-type: none">• Evaluaciones periódicas |
|--|--|---|

En general, los principales retos en el desarrollo de sistemas inteligentes son la identificación del problema, el cual debe tener un componente cognitivo y deben existir recursos para resolver el problema, y la adquisición del conocimiento, en el que se deben recolectar y organizar las fuentes de conocimiento (Bielawski & Lewand, 1991). El componente cognitivo no es más que la necesidad de buscar una solución a través del razonamiento, mientras que la existencia de recursos que permitan encontrar una solución depende de la existencia de al menos un experto en el área o dominio del problema.

Ventajas de los Shells y los productos de Sistemas Inteligentes

A diferencia de la gran variedad de dominios que existen del conocimiento humano, sólo hay una pequeña cantidad de métodos de la IA que se consideran útiles en el desarrollo de sistemas expertos, es decir para representar el conocimiento, para hacer inferencias o para generar explicaciones. Los sistemas construidos con estos métodos útiles sin ningún conocimiento específico del dominio de aplicación se conocen como **shells** (conchas), sistemas esqueléticos o simplemente herramientas de IA.

Construir sistemas expertos haciendo uso de shells ofrece ventajas significativas, ya que un sistema que deba realizar una tarea única puede contar con la integración de un shell que contenga todos los conocimientos necesarios acerca de un dominio de aplicación. Este conocimiento es aplicado por el motor de inferencia para las tareas específicas del dominio de aplicación. Otra ventaja es que el conocimiento representado dentro del sistema puede ser analizado y modificado por un experto si este cuenta con algún tipo de formación en el uso de una shell.

Existe una gran disponibilidad de shells comerciales que pueden variar en tamaño y precios y pueden ser implementados tanto en computadoras de escritorio como en grandes servidores. De igual manera, varía la complejidad que pueden ser sistemas simples basados en reglas o tan complejos que sólo ingenieros del conocimiento pueden utilizarlos satisfactoriamente.

Sin embargo, es conveniente aclarar que los shells no ayudan a la adquisición de conocimiento, pues esto se refiere a la tarea de proporcionarle conocimientos al sistema experto, la cual es realizada por los ingenieros del conocimiento. Por ello, proveer de conocimiento de alta calidad es aún más importante que la elección de un shell, pues el conocimiento del dominio de aplicación definirá el poder de sistema experto.

Alternativas de desarrollo de Sistemas Inteligentes

El desarrollo de sistemas inteligentes que aporten soluciones a problemas de ingeniería y ciencias físicas, biológicas y computacionales, ha permitido que surja la necesidad de utilizar métodos más sofisticados. La rama de la IA que se encarga de estudiar técnicas para la resolución de problemas en esas áreas es la Inteligencia Computacional (IC).

La IC comprende una gama de técnicas que permiten un modelo aproximado de un sistema real debido a que la información es inexacta, no está completa o presenta incertidumbre en sus condiciones, es decir problemas complejos del mundo real que son difíciles o imposibles de analizar desde el punto de vista matemático. En este sentido, es preciso mencionar el concepto de soft-computing que constituye la herramienta principal en esta rama de la IA.

El soft-computing consiste en una variedad de paradigmas de computación que busca incorporar el conocimiento humano de manera eficiente, manejar la imprecisión y la incertidumbre y el aprender a adaptarse a ambientes desconocidos o cambiantes para la obtención de un mejor desempeño (Hernández López, 2008). En otras palabras, esta herramienta utiliza como patrón la mente humana.

Existen cuatro técnicas principales en la utilización del soft-computing (Abraham, 2004):

- 1. Redes neuronales:** Es un modelo de redes computacionales que imita de forma abstracta los mecanismos neurológicos del cerebro, por lo que se asemeja al sistema nervioso central humano.
- 2. Sistemas difusos:** Es un modelo que intenta imitar el nivel cognitivo del ser humano a través del razonamiento. Se basa en la lógica difusa en la que se hace un razonamiento aproximado y adquiere el conocimiento a través de los algoritmos IF-THEN.

3. **Razonamiento probabilístico:** Son métodos que permite evaluar diferentes posibles soluciones de sistemas definidos por la aleatoriedad. Son capaces de actualizar las evaluaciones de resultados anteriores de acuerdo con datos obtenidos más recientemente.
4. **Algoritmos evolutivos:** Son métodos estocásticos o aleatorios que emulan el proceso evolutivo de los seres vivos y su aplicación se ha hecho en problemas de búsqueda, optimización y aprendizaje en máquina.

Cuando hay una sinergia de técnicas a través del soft-computing entonces los sistemas se denominan **sistemas inteligentes híbridos**. En estos la integración de técnicas permite una complete las deficiencias de la otra, lo que además amplía la capacidad del sistema para obtener información y dar solución a problemas complejos que usualmente no pueden ser abordados a través de la utilización de una sola técnica (Osorio & Vieira, 1999).

Aplicaciones de los Sistemas Inteligentes y la IA

La IA tiene diversas áreas de aplicación que se pueden clasificar en dos categorías:

1. **Contenido:** La IA investiga diferentes subcampos que tratan aspectos particulares de la inteligencia, los cuales se generan a partir de la extensa variedad de capacidades complejas y difíciles de modelar o explicar, con la que cuentan los seres humanos, los animales e incluso dispositivos. Los principales subcampos que se pueden mencionar son:
 - **Percepción:** Visión, percepción auditiva y táctil y más recientemente, el gusto y el olfato.
 - **Procesamiento de lenguaje natural:** Producción e interpretación de la lengua hablada y escrita, sea esta manuscrita, impresa o electrónica.
 - **Aprendizaje y desarrollo:** Utilización de redes neuronales.
 - **Planificación:** Solución de problemas, diseño automático a partir de un problema complejo y una colección de recursos, restricciones y criterios de evaluación.
 - **Variedad de razonamiento:** Estudio tanto del razonamiento informal de sentido común como de razonamiento experto especializado.

- **Estudio de las representaciones:** Investigación de las propiedades formales de los diferentes tipos de representaciones, numérica y continua, en contraposición a la estructural y discreta.
- **Técnicas y mecanismos de memoria:** Análisis de las necesidades de los diversos tipos de memoria, incluyendo grandes almacenes de conocimiento.
- **Sistemas multiagente:** Estudio de los diversos tipos de comunicación (lingüística y no lingüística, explícita e implícita, intencional y no intencional).
- **Mecanismos afectivos:** Teoría general que tendría que representar a una amplia variedad de estados afectivos y procesos, incluidos deseos, preferencias, antipatías, placeres, dolores, objetivos de largo plazo, intenciones, ideales, valores, actitudes, estados de ánimo, y mucho más.
- **Robótica:** Es uno de los subcampos más antiguos de la IA y muchas veces se estudia con el propósito de producir nuevos tipos de máquinas útiles y porque diseñar robots de trabajo proporciona un banco de pruebas para la integración de las teorías y técnicas de distintos subcampos de la IA.
- **Búsqueda:** Para solución de algún problema en un espacio de posibilidades, este es un tema recurrente en la IA.
- **Ontologías:** Las formas de representación que utiliza un sistema inteligente.

2. **Técnicas:** Las diversas aplicaciones de la IA han demostrado la necesidad de agrupar sus subcampos en torno a las técnicas de acuerdo con cada clase de problemas. A continuación, se detalla una lista de los subcampos más importantes:

- **Medicina:** Interpretación de imágenes médicas y diagnóstico.
- **Robótica:** Visión, control de motores, aprendizaje, planificación, comunicación lingüística, comportamiento cooperativo.
- **Diversos aspectos de la ingeniería:** Diagnóstico de fallos, sistemas inteligentes de control y de fabricación.
- **Interfaces y sistemas de “ayuda”:** Para las aplicaciones que implican la interacción con los seres humanos.
- **Educación:** Tutores inteligentes y sistemas de gestión de estudiantes.
- **Gestión de la información:** Minería de datos, rastreo web, filtrado de correo, etc.

- **Matemáticas:** Diseño de herramientas para ayudar con distintas clases de funciones matemáticas.
- **Industria del entretenimiento:** Juegos por computadora y sistemas de control y de generación de caracteres sintéticos.
- **Biología:** En el desarrollo de sistemas informáticos más o menos inteligentes para resolver problemas complicados tales como los análisis de ADN.
- **Leyes:** Sistemas expertos para ayudar a los abogados o sistemas para dar asesoramiento jurídico y ayuda a los no letrados.
- **Comercio:** Comercio electrónico gracias al Internet y uso de agentes de software de distintas clases.
- **Espacio:** Control a distancia de los vehículos espaciales y robots autónomos.
- **Actividades militares:** Ámbito en el que se ha gastado la mayor parte de los fondos y dónde no es fácil aprender de los detalles.

El campo de investigación más destacado dentro de la IA son los sistemas expertos (SE). Un sistema experto es una aplicación informática que utiliza conocimiento y procedimientos inferenciales para solucionar problemas con la complejidad suficiente para que sean resueltos por la competencia de un experto humano (Edward A Feigenbaum, 1982). Es decir, es un programa desarrollado para resolver problemas complejos en un dominio particular que requiere de la inteligencia y experiencia humana para su resolución. Estos sistemas se caracterizan porque tienen un alto rendimiento, son comprensibles y confiables por su gran capacidad de respuesta.

Los sistemas expertos tienen tres componentes:

- **Base de conocimiento:** Contiene la información sobre un área específica o dominio, la cual se obtiene de uno o más expertos y contiene las reglas y hechos necesarios para comprender, formular y resolver problemas.
- **Motor de inferencia:** Se considera el cerebro del sistema experto, ya que se encarga de interpretar las reglas y obtener conclusiones con base en los datos.
- **Interfaz de usuario:** Es el componente que vincula el sistema experto con el usuario, ya que le muestra y le permite obtener la información de manera amigable. Además, es el mecanismo para que el motor de inferencia obtenga

datos del usuario cuando no se pueden sacar conclusiones con la información en la base de conocimiento.

La Figura 1 muestra la estructura de un sistema experto, donde se observa la interacción de la base de conocimiento, el motor de inferencia y la interfaz de usuario con otros elementos que hacen que el sistema experto funcione.

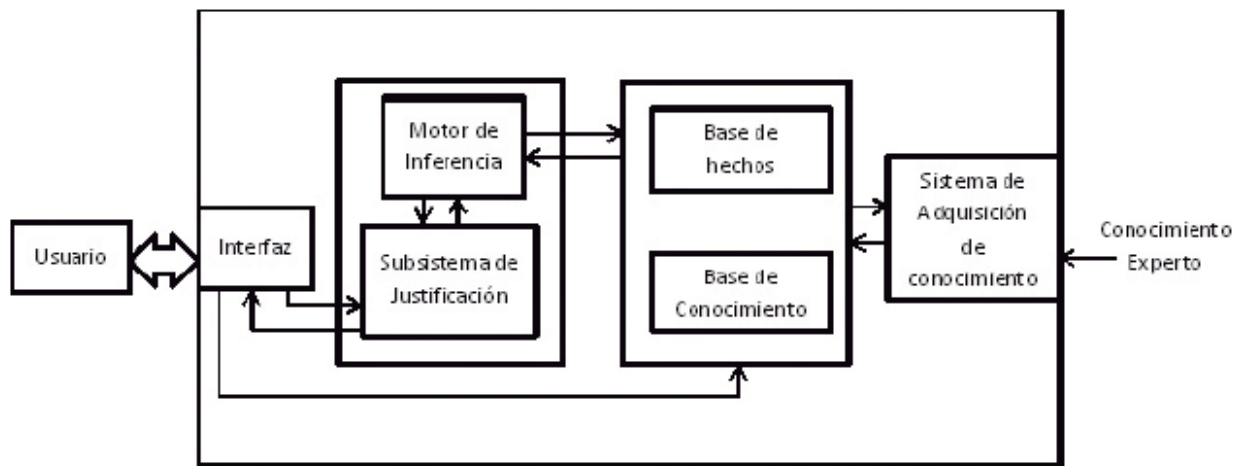


Figura 1. Estructura de un sistema experto. - Tomado de (Badaró, Ibáñez, & Agüero, 2013).

Aunque los sistemas expertos se desarrollen con el propósito de dar solución a problemas, es preciso enumerar las capacidades y limitaciones de estos tipos de sistemas:

Capacidades

- ✓ Asesoramiento
- ✓ Instrucción y ayuda a humanos en la toma de decisiones
- ✓ Demostración (teoremas u otros)
- ✓ Diagnóstico (médico u otros)
- ✓ Razonamiento
- ✓ Interpretación de entrada
- ✓ Predicción de resultados
- ✓ Justificación de una conclusión
- ✓ Sugerencia de opciones a un problema

Limitaciones

- ✓ Sustitución de personas encargadas de tomar decisiones
- ✓ Poseer capacidades humanas
- ✓ Generación de una salida precisa utilizando un conocimiento insuficiente
- ✓ Mejora de su propio conocimiento

Guía evaluativa de herramientas para crear Sistemas Inteligentes

A lo largo de los años se han desarrollado una gran variedad de herramientas para la creación de sistemas inteligentes. Por ello surge la necesidad de utilizar un método de evaluación para determinar cuáles son las herramientas adecuadas y así seleccionar la herramienta que mejor se adapte a los requerimientos de los procesos que el sistema inteligente va a realizar. La siguiente guía pretende orientar en la evaluación de esas herramientas.

Estructura de la evaluación

Se deben tomar en cuenta diversos aspectos relacionados al dominio y al mismo problema que se va a resolver. La estructura de la evaluación propuesta por (Rodríguez R., Hernández C., & Plácido C., 2006) consiste en cinco elementos que se enumeran a continuación:

1. Características de la aplicación
2. Capacidades de la herramienta
3. Métricas
4. Técnicas de evaluación
5. Contextos

Características de la aplicación

Representa el contexto y el dominio tanto del problema como del proyecto. Hay una serie de aspectos que se deben considerar y tendrá mayor significado si se determinan la mayor cantidad de estas características.

- **Características del problema**

- ✓ Dominio del problema: Tipo de conocimiento y restricciones
- ✓ Problema a resolver dentro del dominio: Tipos especiales de conocimiento y procesamiento, restricciones del sistema definitivo (capacidad, rendimiento, fiabilidad, disponibilidad), otros atributos del problema tales como requisitos de almacenamiento, de complejidad y operacionales.
- ✓ Adquisición del conocimiento: Características y restricciones aplicables a las fuentes de conocimiento.

- ✓ Entorno objeto: Hardware necesario, restricciones y requisitos del sistema objeto, características de los usuarios finales.
- **Características del proyecto**
 - ✓ Extensión: Metas, presupuesto y fases en las que se centrará el mismo tales como prototipado, construcción del sistema o adquisición del conocimiento.
 - ✓ Entorno de desarrollo: Software y hardware necesario para que se pueda utilizar la herramienta de desarrollo.
 - ✓ Equipo de desarrollo: Tamaño del equipo para determinar si se requiere o no de una herramienta con capacidad de desarrollo cooperativo, experiencia y preferencias del equipo, características del equipo de ingeniería del conocimiento.

Capacidades de la herramienta

Comprende la funcionalidad de la herramienta que se está considerando utilizar y es preciso centrarse en las capacidades más que en las características que las satisfacen. El siguiente listado muestra algunas de las capacidades y características que las satisfacen (en el mismo orden), que se pueden considerar:

- **Capacidades:**

1. Adquisición del conocimiento	2. Chequeo de consistencia
3. Inferencia y control	4. Manejo de incertidumbre
5. Integración	6. Optimización
- **Características:**
 1. Inducción de reglas, ayudas para construir modelos
 2. Chequeo de la sintaxis de la base de conocimiento
 3. Iteración, encadenamiento hacia adelante o hacia atrás
 4. Factores de certeza, lógica difusa
 5. Acceso a otros lenguajes
 6. Anticipación inteligente, compilación de reglas

Métricas

Son medidas aplicadas para evaluar las capacidades de la herramienta utilizando las técnicas de evaluación que se describirán más adelante. Permiten analizar de manera

más amplia a la herramienta, ya se enfocan en analizar diferentes aspectos de la herramienta. La siguiente lista muestra las métricas que se deben evaluar junto con una breve descripción de cada una, las cuales fueron propuestas por (Rodríguez R. et al., 2006):

- **Claridad:** Facilidad de comprensión y utilización de la herramienta, facilidad de aprendizaje de la herramienta, posibilidad de mantenimiento, coherencia en las características de la herramienta. Es importante aplicar esta métrica durante todas las fases del ciclo de vida del sistema.
- **Coste:** Gastos como costes de integración y entrenamiento, precio de compra y mantenimiento de la herramienta. Puede definirse en términos de dinero, personas, tiempo, entre otros. Durante el ciclo de vida del sistema. Esta métrica se aplica cuando pasa de una fase a otra.
- **Eficiencia:** Velocidad de respuesta, utilización de recursos computacionales y de memoria, velocidad de compilación, tiempo de respuesta, requisitos de memoria de la base de conocimiento. Es de mayor importancia durante la fase operacional del sistema.
- **Extensibilidad:** Facilidad de integración, portabilidad, posibilidades de ampliación. Su importancia aumenta durante el diseño y la implementación del sistema, así como en el mantenimiento si se llegan a requerir nuevas integraciones o ampliaciones.
- **Flexibilidad:** Estructuras de datos, mecanismos de razonamiento, nivel de conveniencia para resolver el problema, sofisticación.
- **Servicio post-venta:** Disponibilidad del sistema, fiabilidad, portabilidad y consistencia de este. Es importante durante la fase de implementación del sistema que es cuando el sistema se distribuye a los usuarios finales del sistema.

Técnicas de evaluación

Las técnicas de evaluación no son más que la manera en que se aplican las métricas listadas anteriormente. Estas técnicas son las siguientes:

- **Benchmarks:** Consiste en formular un problema que funcione como modelo o test. Los test o pruebas pueden ser pequeños o complejos, los más complejos

pueden llegar incluir información acerca del rendimiento y estadísticas. Lo más importantes es que los benchmarks sean específicos.

- **Comparación directa:** No es más que hacer una comparación entre herramientas, donde se estudian sus capacidades y se determina de qué manera estas son diferentes.
- **Técnicas adicionales:** Incluyen entrevistas y cuestionarios, consejos de colegas, estudios de casos, intentos de desarrollo de otros sistemas expertos, sistemas basados en el conocimiento para la evaluación de herramientas.

Contextos

Se refiere a los requisitos necesarios para hacer uso de la herramienta en las diferentes fases del desarrollo del sistema experto. Se mencionan los siguientes:

- **Conceptualización:** Posibilidad que tiene la herramienta para la conceptualización, formalización y descomposición de un problema, además de la identificación y organización de conceptos claves y definición del entorno del problema.
- **Prototipado:** Viabilidad que presenta la herramienta para que el desarrollo sea rápido, de modo que se puedan hacer varias representaciones y posibles implementaciones de forma rápida.
- **Desarrollo:** Facilidades que ofrece la herramienta en cuanto al desarrollo del software.
- **Instalación:** Facilidades que presenta la herramienta cuando se va a instalar el sistema de manera definitiva en el entorno de trabajo.
- **Operación y mantenimiento:** Lo que es capaz de hacer la herramienta para mejorar el rendimiento y las posibilidades de mantenimiento.

Metodología

El objetivo de la metodología es prevenir prejuicios al momento de validar una herramienta cuando se utiliza la estructura propuesta anteriormente. El siguiente y último punto de este capítulo describirá los pasos que componen esta metodología, que fueron propuestos por (Rodríguez R. et al., 2006).

Pasos para evaluar herramientas para crear Sistemas Inteligentes

Existen ocho pasos para la evaluación de herramientas para crear sistemas inteligentes.

Paso 1: Determinar las características de la aplicación

Las propiedades de la aplicación deben ser evaluadas lo antes posible porque permiten inferir las capacidades de la herramienta. Además, determinar las características del proyecto como el alcance, el presupuesto y el equipo de desarrollo ayudarán a establecer requerimientos funcionales y no funcionales.

Paso 2: Identificar los contextos relevantes

Este paso brinda otro factor importante para constituir las capacidades de la herramienta.

Paso 3: Derivar las capacidades significativas de la herramienta

Los dos pasos anteriores derivan las capacidades de la herramienta, por ellos es importante filtrar las capacidades y determinar cuáles son las más relevantes.

Paso 4: Identificar las métricas discriminantes y las técnicas de evaluación

Consiste en establecer cuáles métricas son las que definirán la escogencia de una herramienta. También se deben determinar las técnicas que se utilizarán para aplicar las métricas escogidas.

Paso 5: Identificar las herramientas disponibles

Consiste en encontrar todas las herramientas existentes para el desarrollo del sistema, aunque es difícil encontrar todas las herramientas disponibles.

Paso 6: Filtrar las herramientas disponibles para identificar a las herramientas candidatas

Se filtran las herramientas disponibles que cuentan con los requisitos necesarios para desarrollar el sistema y las restantes se evalúan más detalladamente.

Paso 7: Podar y priorizar cada una de las dimensiones

Se escogen las dimensiones o elementos propuestos en la estructura de la guía evaluativa que tienen mayor prioridad y se eliminan los que son irrelevantes, a excepción de las técnicas de evaluación cuya prioridad debe depender de su disponibilidad y coste.

Para aplicar este paso es necesario conocer muy bien los detalles de los pasos anteriores.

Paso 8: Aplicar el esquema de la estructura para evaluar y seleccionar herramientas

Las dimensiones escogidas en el paso 7 se aplican a las herramientas restantes obtenidas durante el paso 6. Se utilizan las técnicas de evaluación para evaluar las métricas consideradas como relevantes durante el paso 4 de acuerdo con la herramienta en particular.

Capítulo II: Desarrollo de aplicaciones con Expert System Builder

Builder

La creación de sistemas expertos requiere de un software o herramienta que permita insertar fácilmente el conocimiento de uno o más expertos humanos de modo que se pueda dar solución a distintos problemas en un dominio en particular.

El software permite desarrollar la base de conocimiento el cual, junto al motor de inferencia, servirá como un modo de enseñanza para los usuarios. En este sentido, todo el conocimiento de un experto pasa a ser conocimiento del usuario, quien se educa sobre el dominio del sistema a medida que hace uso de este y, por consiguiente, puede volverse un experto.

¿Qué es Expert System Builder?

Expert System Builder o ES-Builder es una herramienta gratuita que permite la creación de sistemas simples para la enseñanza de inferencias y construcción de bases de conocimiento. Fue desarrollado por McGoo Software con fines principalmente educativos y es una versión web mejorada con el marco de desarrollo AJAX. Esta versión se basa en ES-Builder 3.0 que era una aplicación para Windows.



Figura 2. Interfaz de ES-Builder.

La Figura 2 muestra la interfaz de ES-Builder con un sistema experto y las diferentes opciones como edición del árbol de decisión y búsqueda en el sistema experto, además de la cantidad de nodos que tiene el árbol.

ES-Builder almacena los sistemas expertos en una base de datos (MySQL) y el usuario puede almacenar la información en una cuenta, pues para ingresar requiere de una contraseña y un correo electrónico.

Entre las características del ES-Builder se mencionan:

- Permite la visualización instantánea y vista previa del sistema experto como un sitio web.
- Marca los errores en ramas inválidas del árbol de decisión para asistir al usuario que desarrolla el sistema experto y así brindarle asistencia en la identificación de errores en el árbol.
- El usuario puede hacer copias de seguridad y restauración de su proyecto mediante una interfaz web.
- Permite eliminar ramas enteras en lugar de eliminar las partes no deseadas del árbol de decisión.
- Se puede copiar y pegar ramas enteras para reestructurar el árbol de decisión y ayudar en el desarrollo más eficiente de este.

Esta herramienta permite crear un proyecto completo en formato HTML para publicación electrónica y brinda la posibilidad de:

- Hacer búsquedas en el sistema experto
- Mostrar conclusiones
- Mostrar la base de conocimiento
- Mostrar el árbol y la tabla de decisión
- Listar atributos y valores
- Poner a prueba el sistema experto creado para determinar factores de certeza

Etapas de desarrollo del Expert System Builder

Antes de empezar el desarrollo de un sistema experto en ES-Builder se debe definir el tema y el dominio al que este pertenece, se deben establecer el alcance del tema, es decir, los límites. Posteriormente se procede a crear la especificación, la cual consta de dos pasos:

1. **Identificar:** Definir el tema del sistema experto y los límites del tema.
2. **Crear una especificación:** Este paso se subdivide en:
 - a. **Identificación:** Por qué se necesita el sistema experto y qué problema resolverá. Se debe discutir la solución que se dará y no el problema.
 - b. **Especificación:** Definir la solución, quién lo usará, cómo funcionará, dónde y cuándo se utilizará, cómo se pondrá a disposición y qué hardware requerirá. Además, establecer los objetivos para cuando el proyecto esté terminado.
 - c. **Diseño:** Construir el árbol de decisión en ES-Builder utilizando términos cortos para atributos, valores y conclusiones. Otros detalles pueden ser agregados después.
 - d. **Documentación:** Asegurar la mayor cantidad de información posible para cada paso, ya que una persona no relacionada al proyecto debe poder entender de qué trata el mismo. Debe combinar toda la información sobre la identificación y especificación en un instrumento de documentación en el ES-Builder.
 - e. **Implementación del sistema:** Ingresar toda la documentación interna en el panel de datos en ES-Builder como sea posible. Se debe incluir información como: detalle sobre el tema del sistema, larga definición de atributos, larga definición de los valores, larga definición de conclusiones, notas de atributos e imágenes si se requiere, notas e imágenes sobre los valores si se requiere y notas e imágenes sobre las conclusiones. Este paso también incluye la publicación en la web, toda la información de sistema experto se publica desde el ES-Builder y este exporta todo el HTML a la misma carpeta de publicación y las imágenes se exportan desde la página web y luego las copia de la fuente en la carpeta de publicación.

- f. **Pruebas:** Consiste en hacer un informe de prueba. Solicitar a un mínimo de tres personas que utilicen el sistema experto completado, posteriormente estos deben brindar retroalimentación con relación al sistema y brindar críticas que permitan conocer lo que se debe mejorar. Las pruebas deben resumirse en al menos un párrafo y combinar el informe de prueba en un documento de identificación y especificación.
- g. **Evaluación:** Medir el éxito del sistema experto con los objetivos inicialmente establecidos durante el paso de especificación. Brindar una opinión sobre el éxito de cada objetivo particular y sobre el éxito general y comentar sobre las dificultades más importantes surgidas durante el desarrollo del proyecto.
- h. **Presentación:** Agregar detalles bibliográficos y referencias necesarias para la documentación, guardar la carpeta completa a la ubicación especificada por el profesor (si el sistema es desarrollado en clases por un estudiante) y entregar tareas y hojas de criterios con la Declaración de Autoría completada.

La Figura 3 muestra el árbol de decisión de un sistema experto desarrollado con ES-Builder, en donde A es un atributo, V es un valor y C es una conclusión.

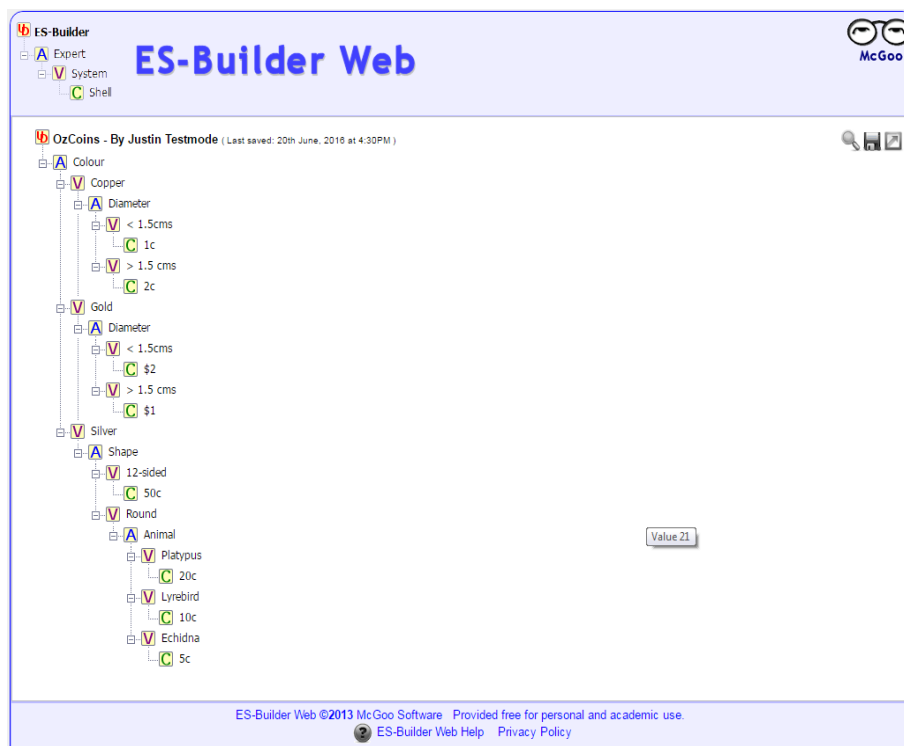


Figura 3. Ejemplo del árbol de decisión de un sistema experto en ES-Builder.

Capítulo III: Desarrollo de aplicaciones con Exsys Corvid

Exsys Corvid, también llamado Corvid, es una poderosa herramienta de propósito general ampliamente utilizada para desarrollar aplicaciones interactivas de sistemas expertos. Está dirigida a aquellas personas que no están familiarizadas con la programación por lo que es de fácil aprendizaje y provee la flexibilidad necesaria para el manejo de resolución de problemas tanto básicos como complejos.

Nació como resultado del trabajo de más de 28 años entre diferentes organizaciones y compañías que buscaban crear una herramienta basada en lo que los desarrolladores necesitan para construir sistemas del mundo real, lo cual corresponde a un enfoque pragmático. Las principales características con las que cuenta esta herramienta son las siguientes (Exsys Corvid, 2011):

- ✓ Su ambiente de trabajo intuitivo permite a los expertos del dominio fácilmente describir de forma lógica los pasos para la toma de decisiones.
- ✓ Utiliza diagramas lógicos de tipo árbol que permiten describir secciones individuales del proceso.
- ✓ Permite la obtención de datos a través de métodos como el encadenamiento hacia adelante y el encadenamiento hacia atrás, lo que posibilita dividir el problema en partes más pequeñas para un rápido desarrollo estructurado.
- ✓ Provee dos tipos de vistas de la lógica: 1) diagramas de árbol que permiten al usuario ver la estructura completa del sistema y 2) un texto de las reglas individuales. El principal beneficio es que, en lugar de ver líneas de código, los usuarios pueden ver la lógica de una manera fácil de comprender que permite a los desarrolladores del sistema editar rápidamente las reglas.
- ✓ El motor de inferencia de Exsys Corvid combina y analiza las reglas para determinar las piezas que se necesitan y cuáles reglas pueden utilizarse para obtener las conclusiones deseadas.

Aunque es fácil aprender a utilizarlo y muchos logran hacer sistemas pequeños en pocas horas gracias a los tutoriales en línea ofrecidos por la propia página de Exsys Corvid, se han desarrollado sistemas altamente complejos a través de esta herramienta, por lo que

es preciso mencionar los tres problemas principales en el desarrollo de sistemas expertos que se resuelven con el uso de Exsys Corvid:

1. Captura de la lógica de toma de decisión: Exsys Corvid provee múltiples formas de describir la lógica a través del uso de reglas IF-THEN basadas en variables, los cuales tienen métodos asociados y propiedades que permiten su uso de diferentes maneras. Estas reglas son muy parecidas al modo en el que un experto las utilizaría para explicar por qué toma una decisión, donde cada regla representa un pequeño paso en la decisión. Un sistema complejo puede tener muchas reglas por lo que Corvid utiliza:

- **Bloques lógicos (Logic Blocks):** Para organizar las reglas y estructurarlas de la misma manera en la que un experto en el dominio pensaría en el problema.
- **Bloques de comando (Command Blocks):** Para describir el flujo de la ejecución del sistema. Permite el uso de los ciclos IF, WHILE y FOR.

2. Construcción de una interfaz de usuario: Exsys Corvid permite que esta sea una tarea simple, por lo que no se requiere de conocimientos en programación en HTML para uso del sistema experto en línea. Para ello, esta herramienta ofrece tres modos con el uso del motor de inferencia Exsys:

- **Ejecución como un applet de Java:** En este las pantallas se definen a través de la configuración de fuentes, colores, posiciones, imágenes, entre otros, lo cual es similar a la edición de un documento en Word. Corvid utilizará la configuración para controlar cómo se harán las preguntas y cómo se presentarán los resultados. En sistemas más complejos se pueden generar reportes en formato HTML, RTF o PDF. También, puede generar todos los archivos necesarios para que el sistema esté en un servidor web.
- **Ejecución como un servlet de Java:** Se aplica en sistemas en donde es necesario un mayor control de la interfaz de usuario y esta se define a través del uso de plantillas HTML, las cuales son ofrecidas por el mismo Corvid y pueden editarse fácilmente con cualquier editor de HTML. Aunque

para esto se necesita un poco de conocimiento en programación con HTML, se pueden crear interfaces de usuario muy avanzadas.

- **Ejecución como un servlet de Java integrado con Adobe Flash:** Se utiliza en sistemas con interfaces de usuario más complejas. Adobe Flash envía datos al Corvid, este los procesa y envía de vuelta datos en formato XML. Este proceso puede repetirse varias veces y permite al Adobe Flash integrarse con el poder de análisis del sistema experto creado con Corvid.

3. Integración con otros recursos de tecnología de la información: Para que un sistema experto funcione de manera eficiente necesita que se integre con otros recursos para obtener datos, guardar resultados o monitorear procesos. Corvid hace que sea fácil conectar el sistema con bases de datos a través del uso de comando SQL y con datos XML a través de comandos XPath. Incluso es posible utilizar código de Java para obtener un mayor nivel de control y añadir cualquier funcionalidad que haga que el sistema experto sea más integral.

Exsys Corvid es una herramienta que permite construir sistemas de manera rápida, gracias a la simplicidad de la sintaxis de las reglas, lo que permite que el experto en el dominio construya el sistema por sí mismo o bien, que trabaje eficientemente con el equipo de desarrollo. Además, ofrece la ventaja de agregar una interfaz de usuario parecido a un sitio web o con un diseño propio, lo que permite que la interacción entre el usuario y el sistema sea similar a la manera en el que una persona obtendría información de un experto. Esto la convierte en una herramienta poderosa con aplicación en una gran variedad de campos de estudio y disciplinas.

Áreas de aplicación de la herramienta

Tanto corporaciones como compañías pequeñas utilizan esta herramienta. Los tipos de sistemas expertos más comunes que se construyen con Corvid son de (Exsys Corvid, 2011):

- Diagnóstico
- Mantenimiento (predictivo o solución de problemas)
- Cumplimiento normativo
- Recomendación de productos

- Atención al cliente
- Análisis de datos

En este sentido, se pueden mencionar las siguientes áreas de aplicación de esta potente herramienta (Exsys Corvid, 2015):

- Agricultura y ambiente
- Apoyo a las decisiones
- Ciencias de la computación
- Diseño web
- Garantía de la calidad
- Geología y minería
- Ingeniería, investigación y desarrollo
- Medicina
- Plantas de operación y control de procesos
- Negocios y comercio electrónico
- Recursos humanos
- Servicios financieros
- Transporte
- Ventas y mercadeo

Construcción de una aplicación con Exsys Corvid

El desarrollo de un sistema experto con esta poderosa herramienta requiere de una serie de pasos básicos (Exsys Corvid, 2011):

- 1. Determinar el problema específico que el sistema va a resolver:** Describir de manera precisa el problema en particular al que el sistema brindará una solución para evitar confusión en pasos posteriores.
- 2. Recolectar la documentación de la lógica del sistema:** Documentar la lógica de la toma de decisiones y los pasos requeridos para resolver el problema, de manera precisa, para que sea más fácil construir el sistema.
- 3. Determinar la arquitectura para el sistema:** Seleccionar aspectos técnicos como el encadenamiento hacia adelante o hacia atrás, determinación de dónde y

cómo manejar operaciones de procedimiento, segmentación de la lógica en bloques reusables, entre otros factores. En general, se requiere que la arquitectura del sistema produzca resultados correctos y sea fácil de entender y mantener por otros.

4. **Utilizar Corvid para describir la lógica del sistema:** Convertir la documentación del paso 2 en reglas utilizando Corvid.
5. **Diseñar la interfaz del usuario final:** Diseñar una interfaz de usuario que produzca los resultados correctos de forma atractiva. El diseño de la interfaz en las primeras fases permite obtener resultados óptimos, ya que se puede probar durante la construcción del sistema.
6. **Probar el sistema:** Hacer pruebas al igual que se hace cuando se desarrolla un proyecto de desarrollo de software. Los resultados deben ser validados por el experto del dominio y las reglas del sistema deben imprimirse para que sean revisadas individualmente por el o los desarrolladores del sistema en Corvid. De igual manera, es importante hacer pruebas con la interfaz de usuario. Estas pruebas pueden hacerse de manera manual que implica correr el sistema varias veces o bien, utilizar la función de validación de Corvid que pone a prueba una gran variedad de casos en un tiempo corto y reporta cualquier tipo de error lógico.
7. **Implementar el sistema:** Una vez validado el sistema, implementarlo en el servidor y ponerlo a disposición de los usuarios finales.

A continuación, se ofrece una guía más resumida de los pasos a seguir para desarrollar sistemas con esta herramienta, los cuales están profundamente basados en los pasos mencionados anteriormente.

Primer paso: Elección de un problema y descomposición en pasos lógicos

Desarrollar un sistema experto en Corvid no sólo requiere determinar el problema al que se le dará solución, pues también es importante tomar en cuenta si al problema se le puede dar una solución a través del uso un sistema basado en el conocimiento. En este sentido, la solución del problema seleccionado debe basarse en el razonamiento lógico y en pasos precisos, por lo que aquellos problemas que requieren de la intuición,

decisiones emocionales y factores al azar no califican para ser resueltos por un sistema experto.

Se debe preguntar “¿Puede el experto explicar a otra persona cómo resolver el problema?”, si la respuesta es “no” entonces no es un buen problema para resolver en Corvid (Exsys Corvid, 2011).

Por otro lado, la descomposición en pasos lógicos no es más que definir la lógica y los pasos que se requieren para resolver el problema, es decir, detallar cómo se llegará a la solución. Esto es muy importante ya que puede facilitar enormemente el proceso de desarrollo del sistema experto en Corvid.

Pantalla de variables

Las variables son los elementos clave en Corvid ya que se utilizan para describir el proceso de toma de decisión, hacer preguntas y mostrar resultados. Al igual que las variables en los lenguajes de programación, estos poseen un identificador y un valor asociado.

Las variables se utilizan para construir las reglas del sistema, pero antes deben crearse en la pantalla de variables que se despliega cuando se va a iniciar un nuevo sistema. En la Figura 4 se señala con una flecha el botón “Variables” que debe seleccionarse para que se despliegue la pantalla de variables.

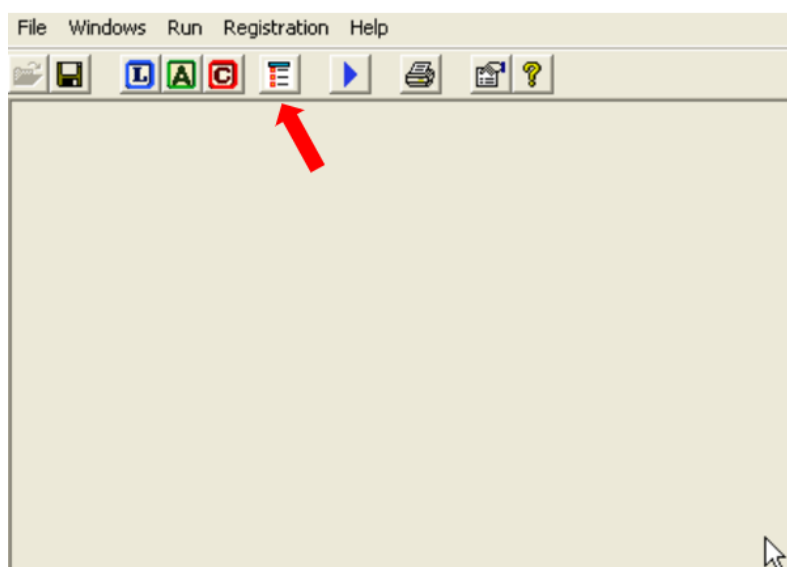


Figura 4. Acceso a la pantalla de variables a través del icono “Variables”.

También se puede acceder a “Variables” haciendo clic en “Windows” en la parte superior izquierda de la pantalla, tal como se muestra en la Figura 5.

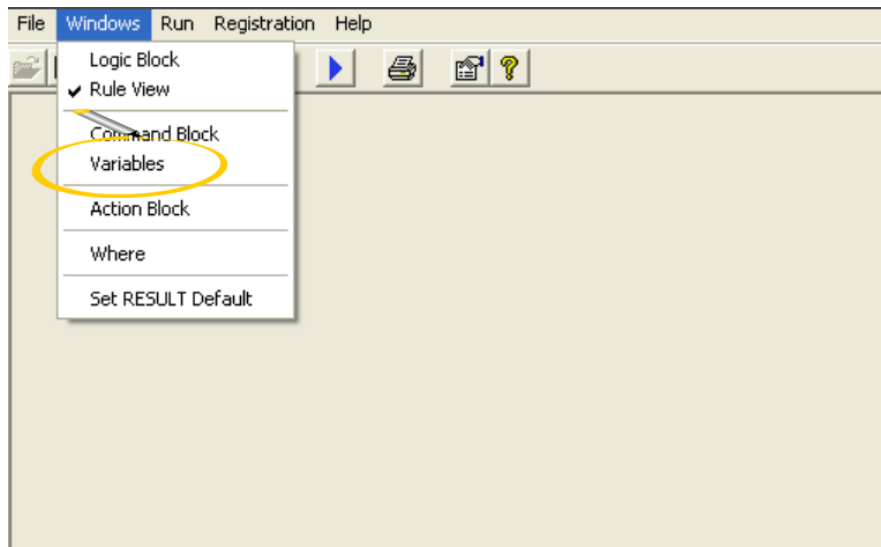


Figura 5. Acceso a la pantalla de variables a través de "Windows".

La pantalla de variables lista las variables en el cuadro de la parte izquierda, mientras que en la derecha permite configurar diferentes propiedades para cada variable como se muestra en la Figura 6.

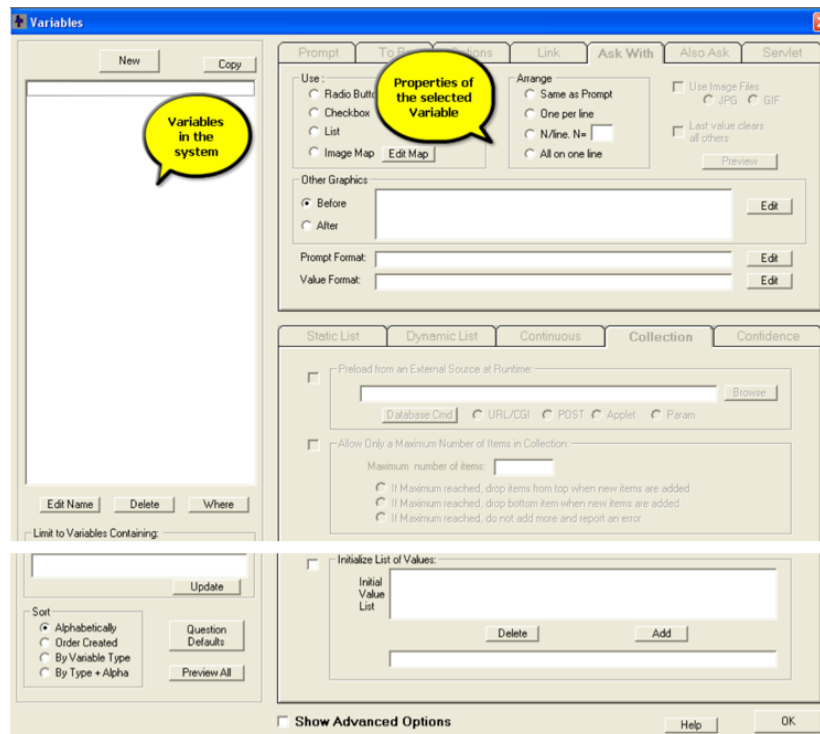


Figura 6. Pantalla de variables.

- **Área de formato de variables**

Esta sección de la pantalla de variables brinda la posibilidad de configurar las propiedades de las variables. La Figura 7 muestra el área donde se le da formato a las variables con diferentes propiedades, como se observa existen diferentes pestañas en la parte superior las cuales se utilizan para que el usuario ingrese el valor de una variable y se muestra un ejemplo del uso de una de estas opciones. Las pestañas que más comúnmente se utilizan son las siguientes:

- ✓ *Prompt*: Permite ingresar una pregunta o frase la cual será mostrada al usuario cuando el sistema le solicite ingresar el valor.
- ✓ *Options*: Permite definir un valor por defecto para la variable. Cuando el sistema le pregunte al usuario por el valor de la variable, el sistema mostrará un valor preseleccionado y el usuario sólo debe aceptar el valor mostrado o bien, cambiarlo.
- ✓ *Ask With*: Permite seleccionar el tipo de control a utilizar cuando se le indique al usuario que debe escoger un valor para la variable. Algunos de los controles son Radio Button, Checkbox y mapa de imágenes.

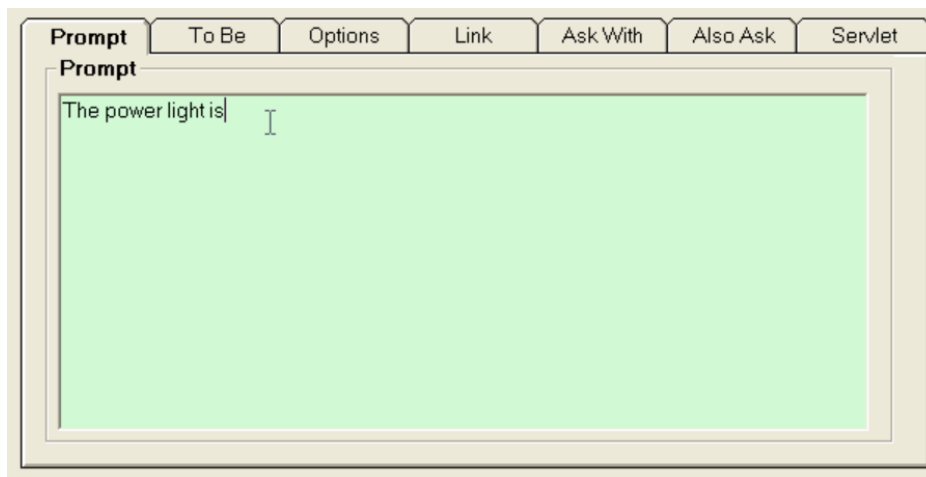


Figura 7. Área de formato de variables.

- **Área de lista de variables**

Las variables que se utilizarán en el sistema aparecen como una lista en el cuadro de la parte izquierda de la pantalla de variables, tal como se muestra en la Figura 8 con un ejemplo de tres variables.

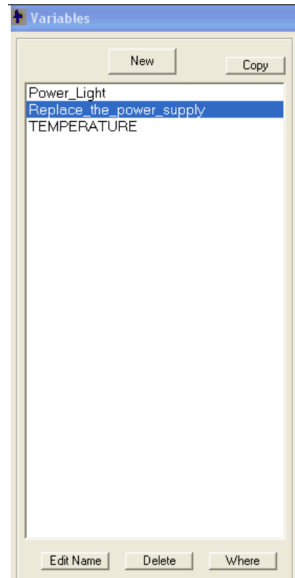


Figura 8. Área de lista de variables.

- **Área de valores de las variables**

Dependiendo del tipo de variable, el valor de esta se puede preguntar al usuario para que ingrese el dato o bien, se puede asignar el valor. También, se le puede proporcionar al usuario las opciones de manera que se delimiten los datos que serán utilizados por el sistema. En la Figura 9 se observa el área de valores de las variables y se muestra un ejemplo de los valores que puede tener una variable de tipo estática. En este caso la variable llamada “Power_Light” puede tener el valor “On”, “Off” o “Blinking”, una de estas opciones será escogida por el usuario. De igual manera, se observa que se pueden editar los valores de otros tipos de variables.

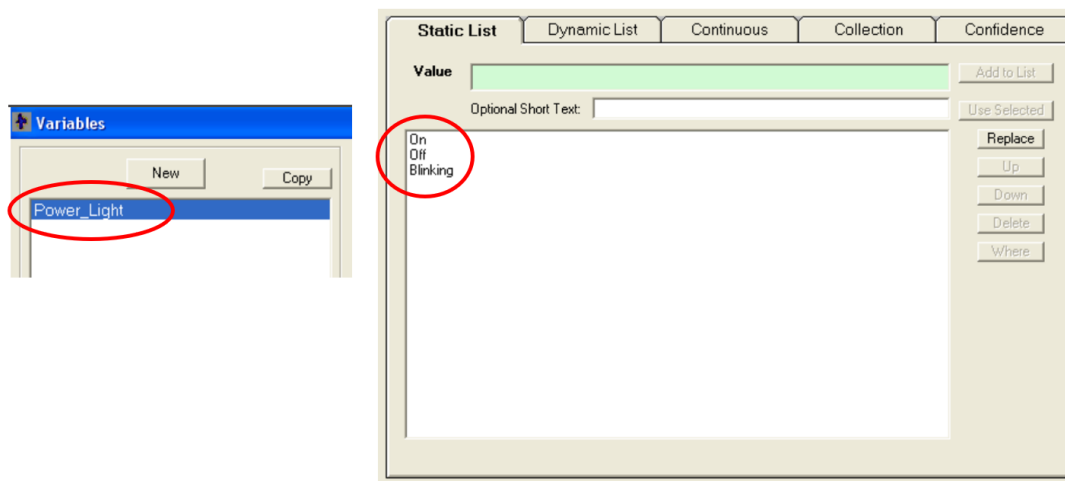


Figura 9. Área de valores de las variables.

- **Añadiendo variables**

Para agregar una nueva variable en el sistema, se debe hacer clic en el botón “New” mostrado en la Figura 10 que se encuentra en la parte superior del área de lista de variables.

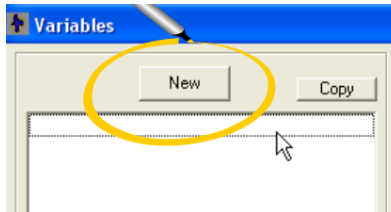


Figura 10. Selección del botón "New".

Luego se desplegará la ventana “New Variable” que aparece en la Figura 11, en donde se le dará un nombre único y se seleccionará el tipo al que corresponde la variable que se va a crear. Se recomienda que el nombre de la variable sea corto y describa su significado.

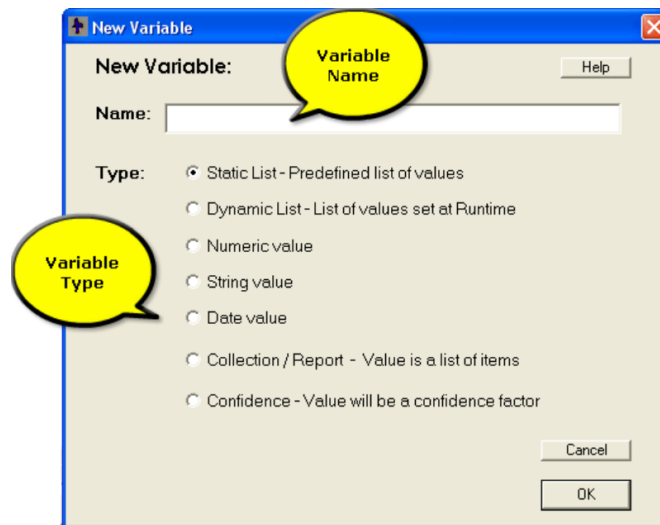


Figura 11. Ventana “New Variable”.

Definiendo valores de las variables Corvid

Corvid tiene siete tipos de variables que se pueden observar en la Figura 11, de las cuales las cinco primeras se utilizan para construir expresiones booleanas que van dentro de las reglas y pueden utilizarse para hacerle preguntas al usuario. La mayoría de los sistemas desarrollados en Corvid están contruidos con variables de tipo estática

y numéricas para definir las reglas de tipo IF, mientras que los dos últimos tipos de variables se utilizan para hacer recomendaciones.

- **Variables de tipo lista estática**

Son variables que tienen una lista específica de múltiples valores y puede haber cualquier cantidad de valores en la lista. Los valores son los mismos para todos los usuarios del sistema y deben utilizarse cuando se requiera una lista fija de los posibles valores. Es el tipo de variable que se utiliza más comúnmente en Corvid.

- **Variables numéricas**

Son variables a las que se les asigna un valor numérico y puede obtenerse preguntándole al usuario, a través del cálculo realizado por las reglas, por medio de un recurso externo, etc.

- **Variables de confianza**

Son variables a las que se les asigna un valor que indica el grado de certeza en un resultado específico o recomendación. El valor de este tipo de variables es numérico y permite medir la probabilidad de que dicha variable se aplique a una situación particular. Este tipo de variables tienen un uso más efectivo en sistemas donde hay muchas recomendaciones posibles.

Definición y representación gráfica

Como se mencionó anteriormente, la ventana “New Variable” ofrece la opción de agregar un tipo de variable denominada **variable de confianza**, la cual permite medir el grado de incertidumbre de la respuesta o solución que el sistema proporcione al problema en cuestión. Por ello, para tener una mejor comprensión de cómo configurar este tipo de variables en el sistema, es conveniente comprenderlo a través de un ejemplo, el cual fue tomado de (Exsys Corvid, 2007).

La Figura 12 muestra la ventana “Variables” en donde se ha añadido al sistema la variable de confianza “Highway” y se ha colocado un “Prompt” con el texto “Go to work via the Highway”, visto en el área de formato de variables.

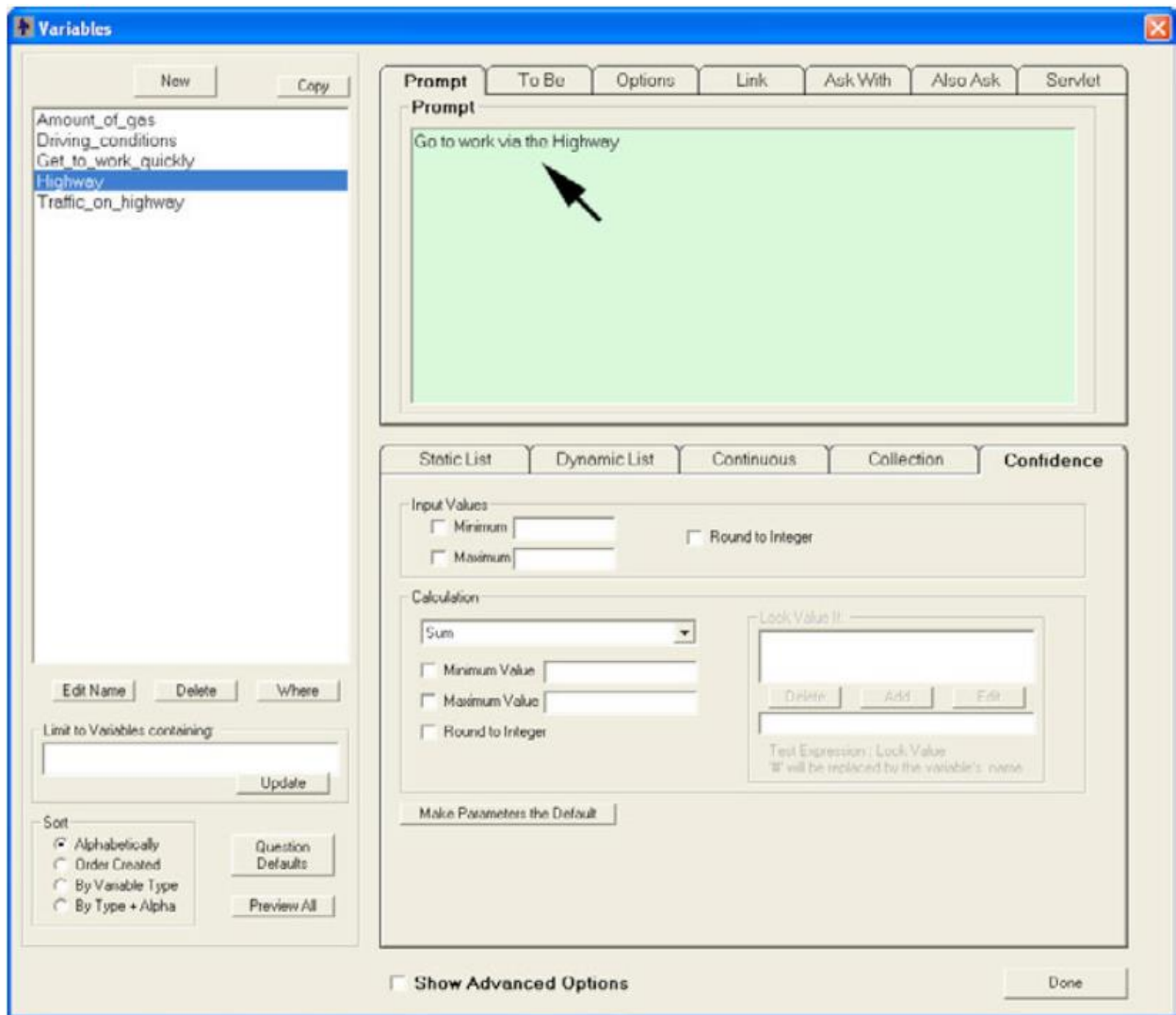


Figura 12. Ejemplo de un sistema donde se utiliza una variable de confianza. - Tomado de (Exsys Corvid, 2007).

Luego, se procede a ir al área de valores de las variables en la mitad inferior de la ventana y se selecciona la pestaña “Confidence”. Esta pestaña mostrará varias opciones que permiten combinar los valores que se le asignarán a la variable. Por defecto, el método “Sum” es el que aparece para calcular este valor, el cual resume los valores que se le asignen a la variable. Siguiendo con el mismo ejemplo de la figura anterior, es necesario añadir otras dos variables de confianza: “Coast_Road” y “City_Streets”. Estas dos nuevas variables deben aparecer en el área de lista de variables, tal como se muestra en la Figura 13.

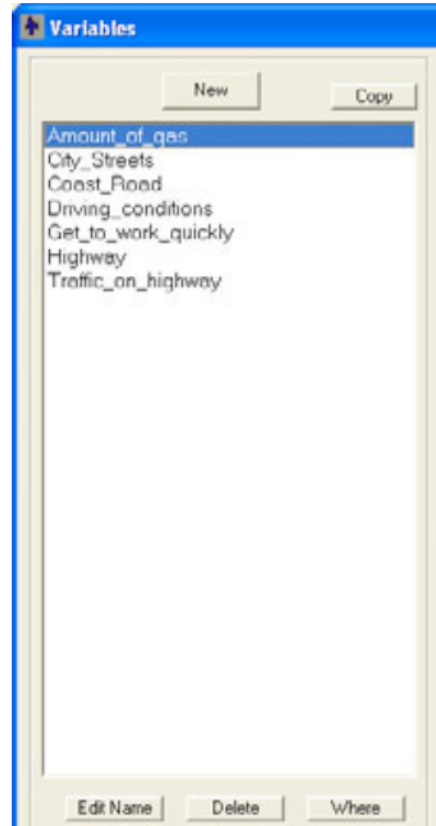


Figura 13. Área de lista de variables donde se muestran las variables de confianza "Coast_Road" y "City_Streets". - Tomado de (Exsys Corvid, 2007).

Finalmente, se puede observar en la Figura 14 la representación gráfica de una parte del sistema, donde se muestra un diagrama de árbol de la toma de decisiones. Se observa la representación de las variables de confianza "Highway" y "Coast_Road" en el ejemplo que se ha explicado en esta sección.

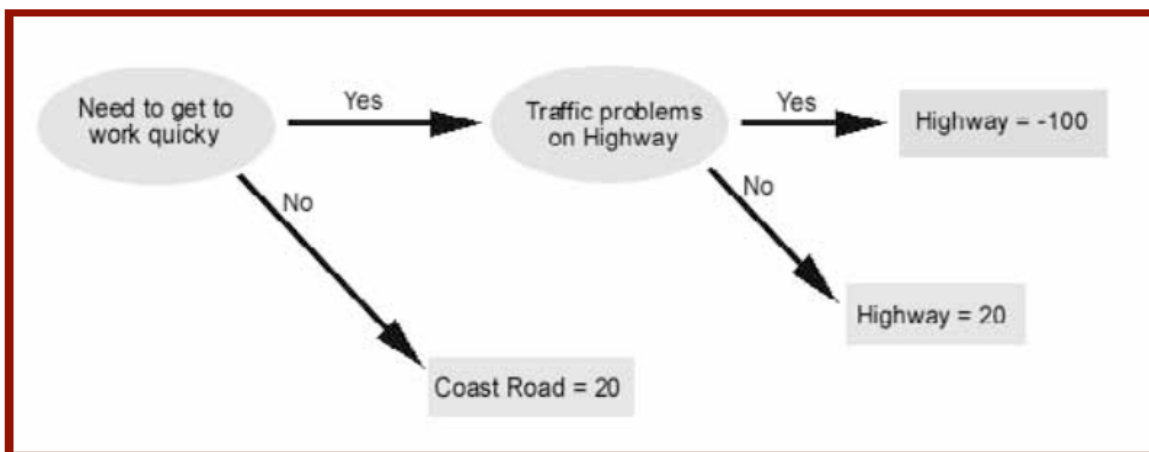


Figura 14. Diagrama de árbol de la toma de decisiones donde se incluyen dos de las variables de confianza añadidas al sistema. - Tomado de (Exsys Corvid, 2007).

La toma de decisiones que el diagrama de árbol representa, es lo que se estructura a través de las reglas IF-THEN. En Corvid estas se representan a través de los denominados **nodos** que son representados a través de los bloques lógicos.

En el caso del ejemplo, tomando en cuenta el diagrama de árbol mostrado en la Figura 14, los bloques lógicos quedarían de la manera mostrada en la Figura 15, donde los paréntesis verdes representan los nodos IF y las flechas amarillas representan los nodos THEN. Se observa que para el caso del ejemplo, los nodos THEN corresponden a las variables de confianza añadidas anteriormente.

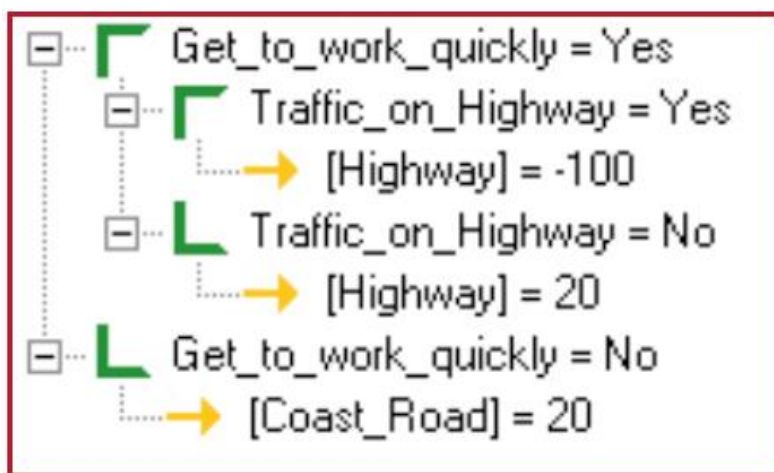


Figura 15. Bloque lógico que representa en Corvid las reglas IF-THEN del sistema. - Tomado de (Exsys Corvid, 2007).

Estos nodos pueden añadirse, pero esto se explicará con mayor detalle más adelante en este capítulo, ya que se requiere de una sección aparte para una mejor descripción.

Ventana de comandos

La ventana de comandos “Display Commands”, que se muestra en la Figura 16, se compone de:

1. El área “Commands” que despliega una lista de los comandos en la parte superior.
2. Bajo el área de comandos se encuentra una pequeña ventana que muestra el comando que se está editando o construyendo.
3. En la parte inferior está una serie de controles que permiten construir y dar formato a los comandos mostrados en el área “Commands”.

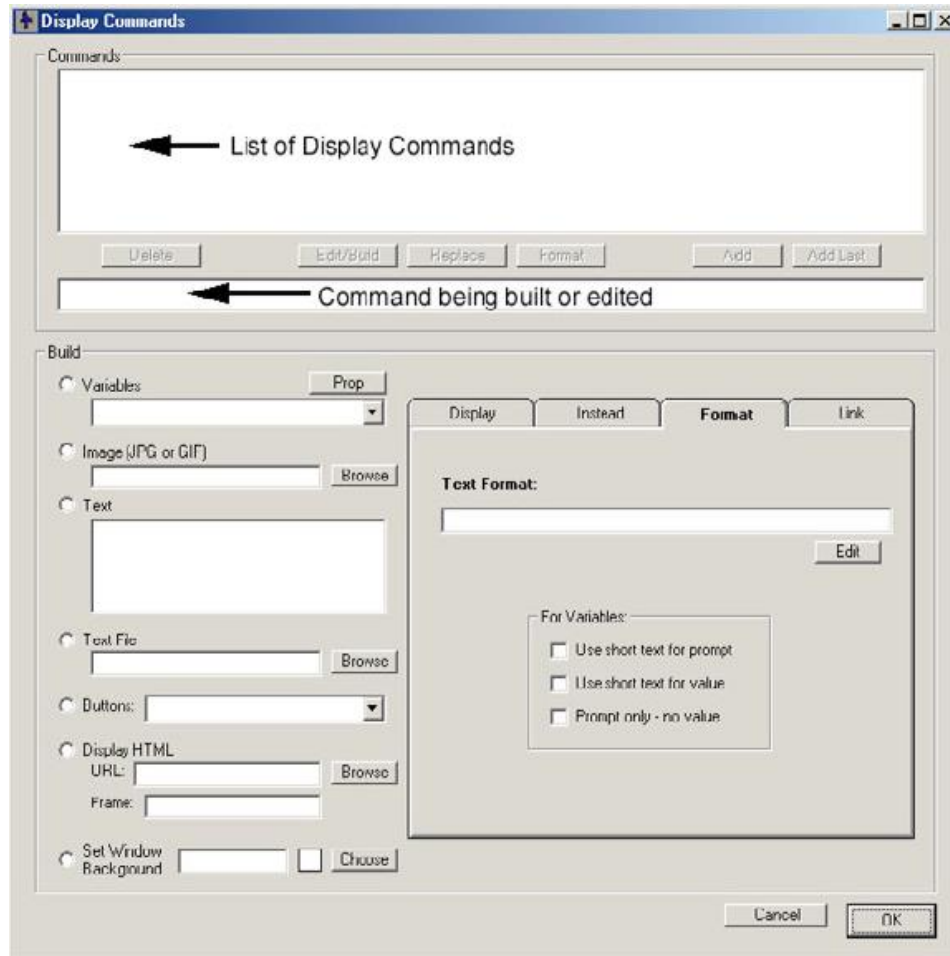


Figura 16. Ventana de comandos "Display Commands". - Tomado de (Exsys Corvid, 2007).

Esta ventana permite construir diferentes tipos de comandos tales como: variables, imágenes, texto, archivos de texto, botones e incluso, mostrar una página web a través de la dirección URL de la página.

Segundo paso: Añadir bloques lógicos

Corvid ofrece una manera única para definir, organizar y estructurar las reglas del sistema en bloques lógicamente relacionados. Estos son grupos de reglas que pueden definirse a través de diagramas de árbol o como reglas individuales. Cada bloque contiene varias reglas y árboles o bien, una sola regla. El bloque contiene reglas que se relacionan con un aspecto específico de la tarea de toma de decisión.

La ventana de bloques lógicos "Logic Block", mostrado en la Figura 18, puede ser accedida haciendo clic en el botón de la Figura 17. Esta ventana provee los controles y

herramientas necesarias para construir y estructurar las reglas de forma que se asemeje al proceso de razonamiento para buscar la solución de un problema.



Figura 17. Botón para acceder a la ventana de bloques lógicos. – Tomado de (Exsys Corvid, 2011).

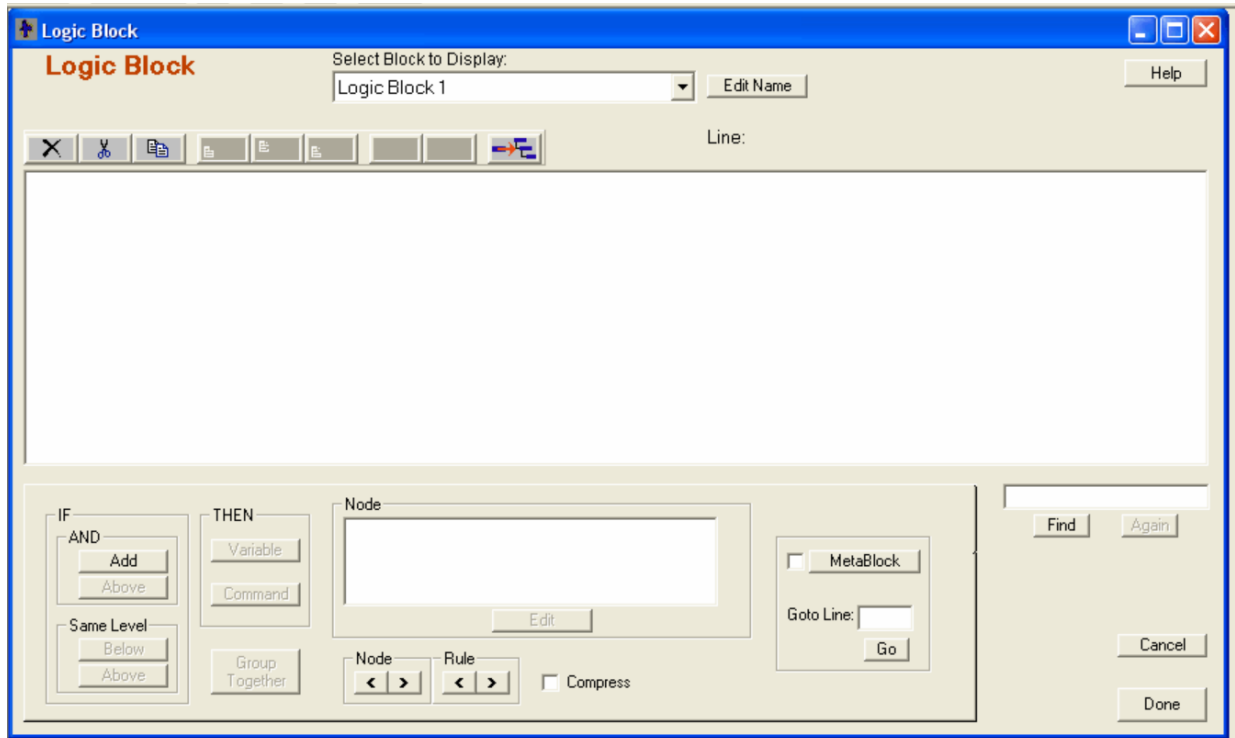


Figura 18. Ventana "Logic Block".

Con base en la Figura 18, en la Figura 19 muestra dónde aparece el nombre del bloque lógico que puede ser modificado por el desarrollador del sistema, mientras que la Figura 20 muestra el área de trabajo. En la parte superior se observan los botones que facilitan cortar, copiar y pegar contenido en la estructura del bloque lógico.

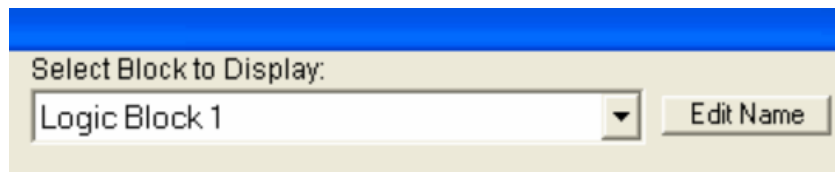


Figura 19. Nombre del bloque lógico.

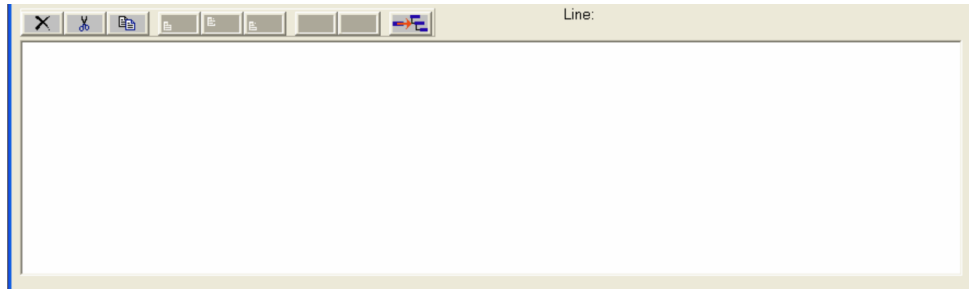


Figura 20. Área de trabajo y botones de cortar, copiar y pegar en la parte superior.

En la Figura 21 se observa el área de nodos que permite obtener detalles del nodo seleccionado.

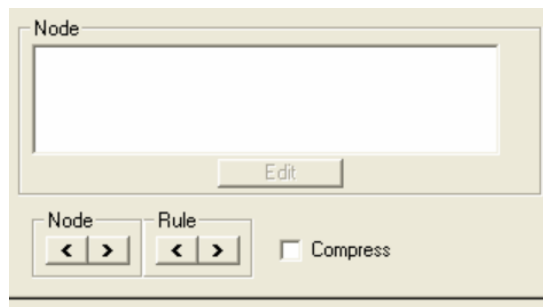


Figura 21. Área de nodo.

El control "MetaBlock", que se puede observar en la Figura 22, permite construir un tipo especial de bloques lógicos que combina la lógica de la regla del bloque con datos de un archivo de una hoja de cálculo.



Figura 22. Control "MetaBlock".

Por último, la Figura 23 muestra el botón "Find" que da la opción de hacer una búsqueda rápida de una cadena de caracteres dentro del bloque lógico.

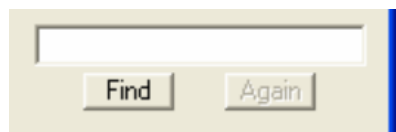


Figura 23. Botón "Find".

Bloques de control

Los bloques de control son los que permiten añadir los bloques lógicos y los nodos para las reglas IF-THEN. Los controles que admiten estas opciones se mostraron anteriormente, por lo que en esta sección se explicará cómo agregarlos al sistema.

- **Añadiendo los bloques lógicos**

Antes de añadir los bloques lógicos es necesario crear las variables que se utilizarán en el sistema experto. Es preciso mencionar que cuando se muestra la ventana Logic Block, también aparece la ventana “Rule View” que se observa en la Figura 24. Esta ventana permite ver el texto completo de una estructura IF-THEN seleccionada, lo que facilita examinar a detalle una regla en específico.

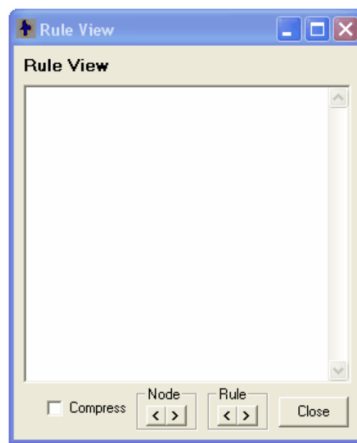


Figura 24. Ventana "Rule View".

En la mitad inferior de la ventana Logic Block están los controles para agregar nodos IF-THEN mostrados en la Figura 25.

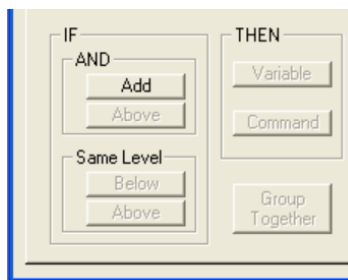


Figura 25. Controles para agregar nodos IF-THEN.

Al hacer clic en el botón “Add” aparece la pantalla “Add To Block”, mostrada en la Figura 26, que muestra las variables en el sistema en el cuadro de la izquierda. También brinda

otras opciones como agregar nuevas variables cuando se requiera u ordenar la lista de variables para facilitar la búsqueda cuando hay una gran cantidad.

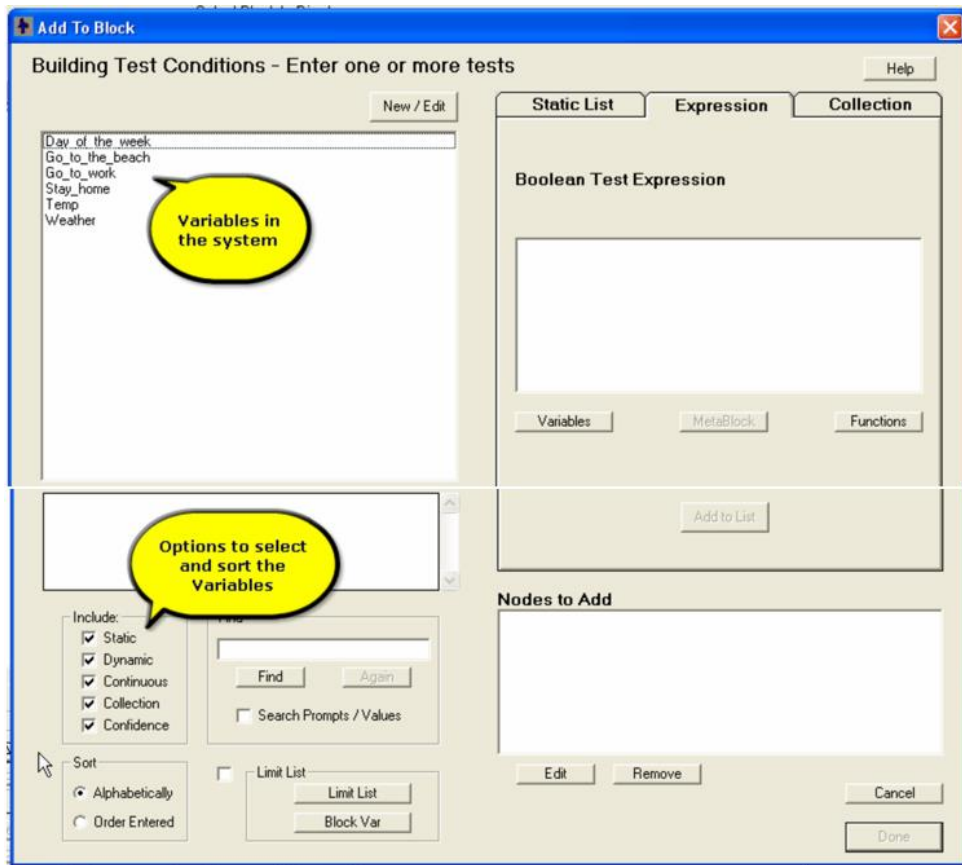


Figura 26. Pantalla "Add To Block".

En la mitad derecha de la pantalla, el cuadro superior se utiliza para construir condiciones individuales que posteriormente serán los nodos del bloque lógico, mientras que el cuadro inferior llamado "Nodes to Add" muestra los nodos que se han construido y que serán agregados al sistema. Para ello sólo se debe hacer clic en el botón "Add to List" y luego en el botón "Done" para que aparezcan en el área de trabajo de la ventana Logic Block, en forma de un diagrama de árbol. Los nodos aparecerán en el formato `NOMBRE_VARIABLE = valor`, o bien, `NOMBRE_VARIABLE = valor1, valor2, ...,` tal como el ejemplo de la Figura 27.

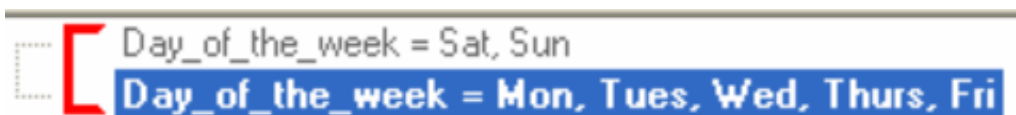


Figura 27. Ejemplo del formato de un nodo en el bloque lógico. – Tomado de (Exsys Corvid, 2011).

- **Añadiendo las condiciones IF**

Las condiciones IF se muestran como un grupo relacionado de nodos que proveen condiciones para una misma variable. En la Figura 28 se observa un ejemplo, donde la variable "COLOR" tiene diferentes valores que servirán como las condiciones de las reglas del sistema. El paréntesis rojo indica un camino que puede servir para agregar otros nodos dentro de alguna de las condiciones, como se observa en el ejemplo de la Figura 29.

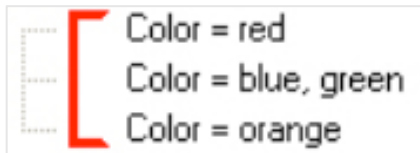


Figura 28. Ejemplo de condiciones IF. – Tomado de (Exsys Corvid, 2011).



Figura 29. Ejemplo de condiciones IF con condiciones dentro de ellas. – Tomado de (Exsys Corvid, 2011).

Es preciso mencionar que las condiciones IF siempre son de tipo booleano por lo que evalúan entre True o False.

- **Adición de nodos ENTONCES**

Los nodos ENTONCES o THEN asignan un valor a una variable y se agregan debajo de cualquiera de las condiciones IF y estos se indican por medio de flechas de color amarillo como se muestra en el ejemplo de la Figura 30. El paréntesis de color verde indica que es un camino bajo el cual hay una regla.

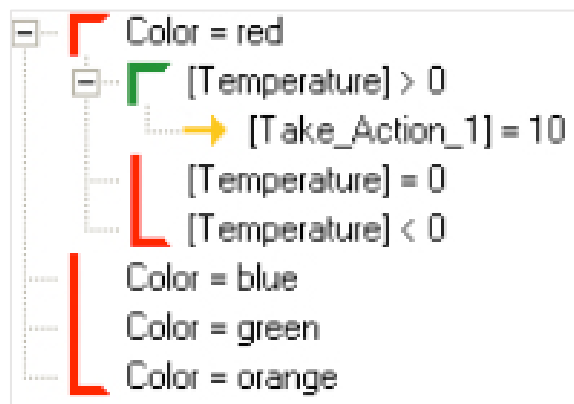


Figura 30. Ejemplo donde se muestra un nodo THEN. – Tomado de (Exsys Corvid, 2011).

Toda condición debe tener un nodo THEN para que se complete una regla, por lo que el paréntesis rojo brinda la facilidad de darse cuenta si hacen falta reglas en el sistema.

Los nodos IF y THEN ayudan a construir la lógica necesaria para la toma de decisiones en el sistema y estas reglas serán utilizadas por el motor de inferencia para dar una solución, ya sea a través del encadenamiento hacia adelante o hacia atrás.

Tercer paso: Añadir bloques de comandos

Un bloque de comandos o “Command Block” en Corvid se encarga de decirle al sistema qué hacer, a diferencia de los bloques lógicos que indican el cómo hacer las cosas. En otras palabras, los bloques de comandos contienen comandos de procedimiento que le indican al motor de inferencia como usar las reglas del sistema. Por ello, es preciso mencionar que todos los sistemas deben tener al menos un bloque de comando y que estos no se utilizan para la lógica de las reglas del sistema.

Se pueden mencionar los siguientes usos más comunes para estos tipos de bloques:

- Determinación del tipo de encadenamiento para la ejecución de las reglas.
- Control del orden en el que se ejecutan los bloques.
- Obtención de datos de programas externos al iniciar la sesión.
- Envío de resultados y recomendaciones del sistema a otros programas.
- Ejecución de un conjunto de reglas varias veces para los ciclos FOR o WHILE.
- Despliegue de ventanas complementarias como ayuda o para propósitos de explicación.
- Visualización de reportes.
- Envío de correos.

Para agregar y editar bloques de comandos se debe hacer clic en el botón “Command Block”, mostrado en la Figura 31, la cual desplegará la ventana que se muestra en la Figura 32.

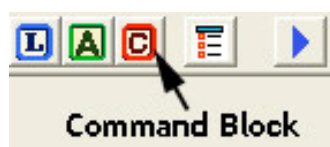


Figura 31. Botón para acceder a la ventana "Command Block". – Tomado de (Exsys Corvid, 2011).

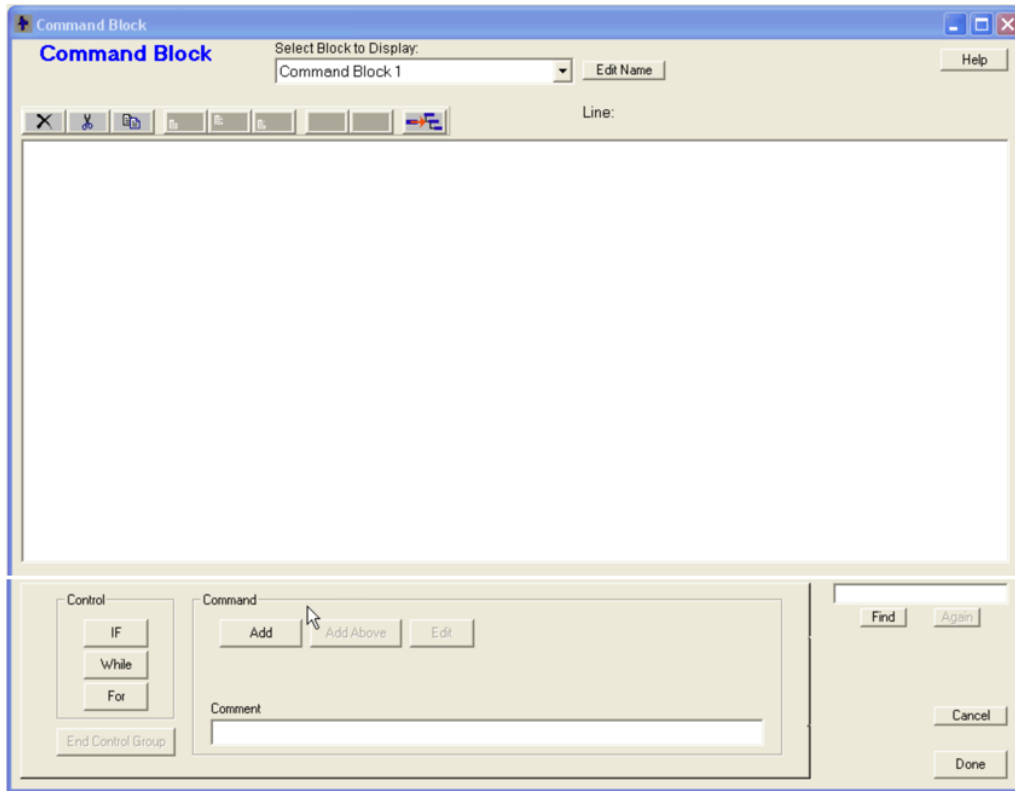


Figura 32. Ventana "Command Block".

La ventana Command Block es similar a la ventana Logic Block ya que también tiene botones para cortar, copiar y pegar los comandos que se coloquen en el área de trabajo. De igual manera, también tiene la barra que muestra el nombre del bloque de comando en la parte superior de la ventana y en la parte inferior derecha se encuentra el botón "Find" para la búsqueda de cadena de caracteres en el área de trabajo.

En la mitad inferior de la ventana se encuentra el grupo de botones "Control" visto en la Figura 33 permiten agregar comandos de bifurcación y bucles, mientras que en la Figura 34 se muestran los botones "Command" que permiten añadir y editar comandos.



Figura 33. Botones "Control".

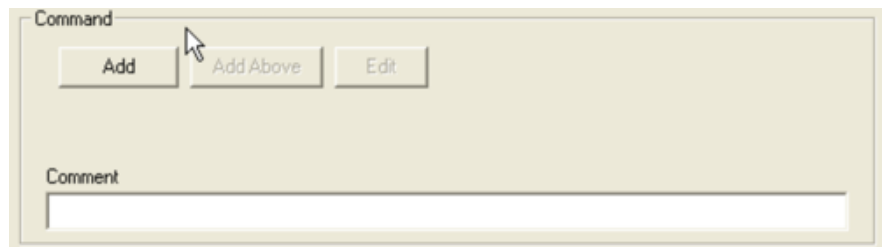


Figura 34. Botones "Command".

Al hacer en el botón “Add” del área de botones Command se desplegará la ventana “Commands” mostrada en la Figura 35 y ayuda a asegurarse que los comandos sean válidos y los convierte automáticamente en la sintaxis correcta para Corvid. Esta ventana tiene varias pestañas y a continuación se describen las que más se utilizan en los sistemas expertos:

- ✓ *Variables*: Se utiliza para construir comandos que: 1) fijan el valor de una variable, 2) infieren el valor de una variable o grupo de variables a través el encadenamiento hacia atrás y 3) causan que al usuario del sistema se le haga una pregunta inmediatamente, sin importar la lógica del sistema.
- ✓ *Blocks*: Provee los comandos necesarios para ejecutar sistemas que utilizan el encadenamiento hacia adelante para la ejecución de los bloques.
- ✓ *Results*: Permite despliega una pantalla de resultados al finalizar la sesión y se puede configurar para que muestre sólo ciertas variables, imágenes y texto con formatos de color, tamaño y fuente.

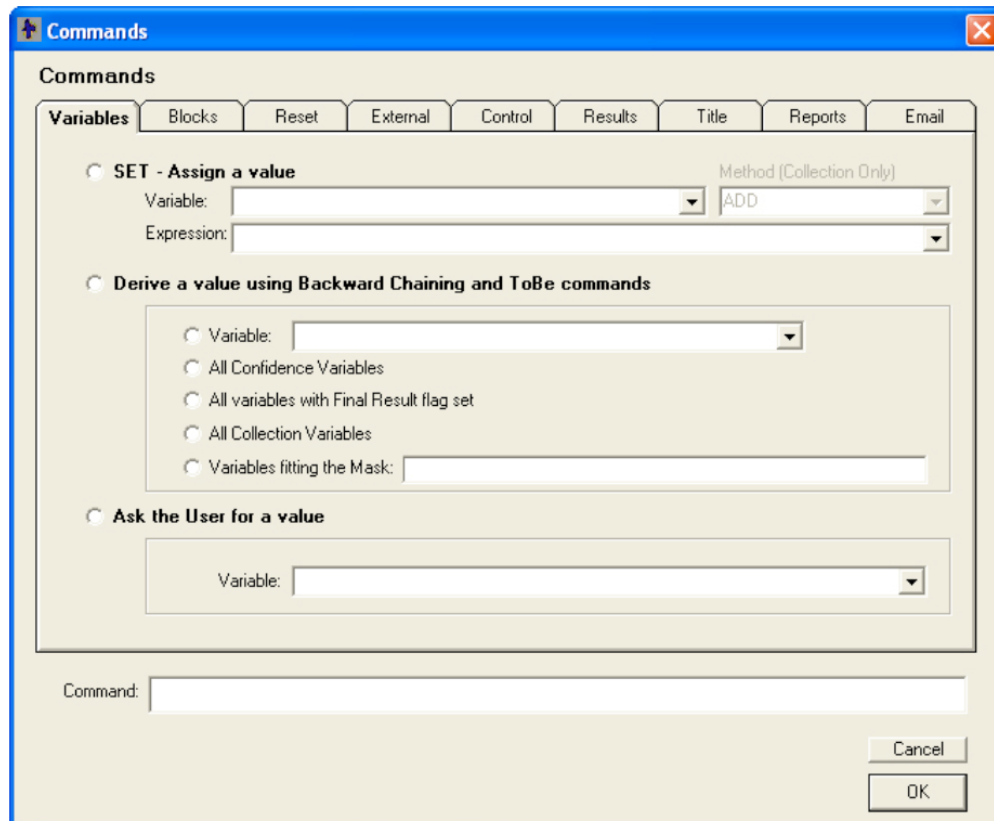


Figura 35. Ventana "Commands".

En la Figura 36 se aprecia un ejemplo de un bloque de comando simple. Muchos de los sistemas desarrollados en Corvid funcionan con bloques de comandos simples que tienen pocos comandos, sin embargo, sistemas más complejos que requieran operaciones de procedimientos específicas o de la integración con otros programas necesitan de bloques de comandos más complejos.



Figura 36. Ejemplo de un bloque de comando simple.

Es preciso mencionar que, si se ejecuta un sistema sin un bloque de comandos, aparecerá la ventana "Build Command File" mostrada en la Figura 37.

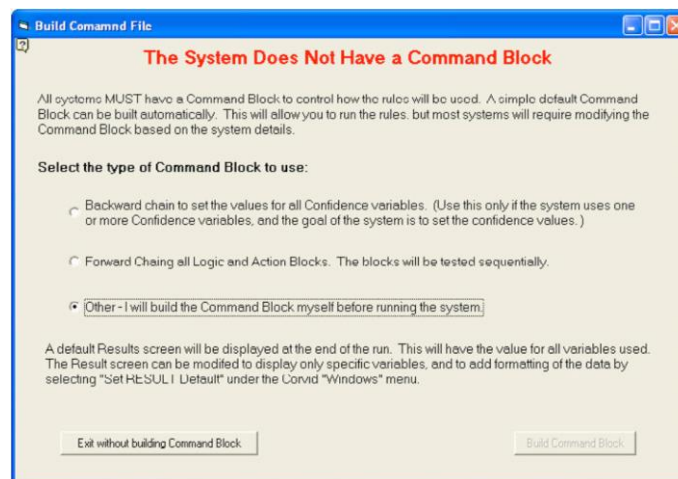


Figura 37. Ventana "Build Command File".

Cuarta etapa: Ejecución

Para ejecutar un sistema desarrollado en Corvid se debe hacer clic en el ícono del botón "Run" mostrado en la Figura 38. Existen varias formas de ejecutar un sistema, pero por defecto Corvid utiliza el Exsys Corvid Applet Runtime que es una forma de ejecutar un programa de Java como parte de una página web.

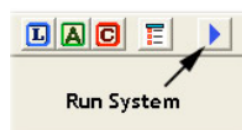


Figura 38. Botón "Run". – Tomado de (Exsys Corvid, 2011).

Cuando se ejecuta el sistema, Corvid crea un archivo CVR el cual es la versión de tiempo de ejecución del sistema creado. Luego Corvid crea una página web predeterminada con el código para introducir el archivo CVR en la página, y los parámetros que hacen que el archivo se cargue y ejecute en el sistema. El sistema se cargará y se ejecutará en una ventana del navegador y los archivos creados por Corvid pueden moverse a cualquier servidor web y el sistema puede ponerse para ser usado en línea.

Como se mencionó, Corvid tiene una plantilla predeterminada para la página web que contiene el archivo CVR y se puede apreciar un ejemplo de esto en la Figura 39. Las preguntas del sistema se despliegan dentro del cuadro blanco y la plantilla puede cambiarse si se desea a través de cualquier editor HTML.

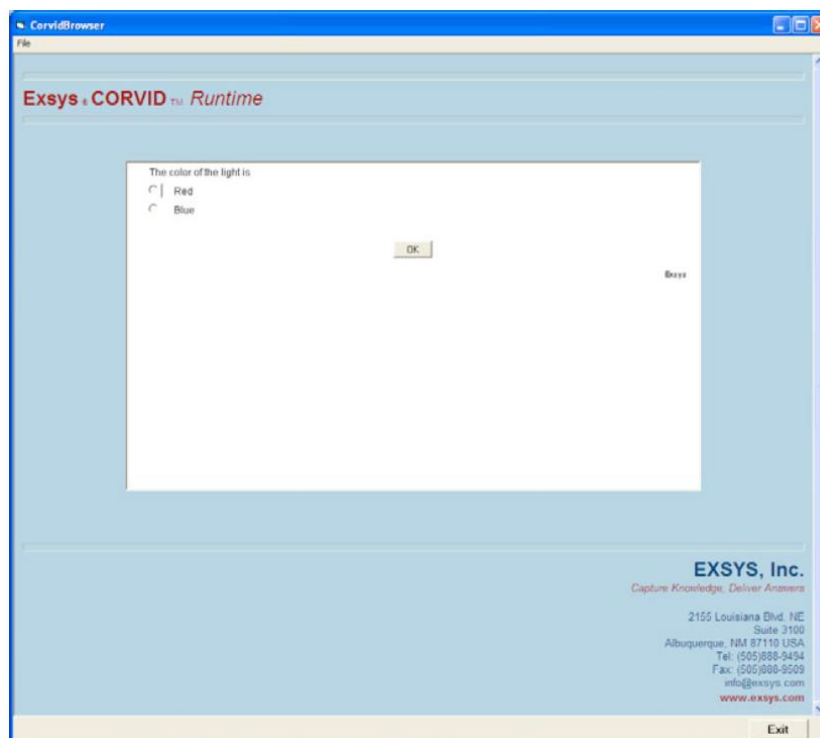


Figura 39. Ejemplo de plantilla predeterminada para la página web de un sistema creado en Corvid.

El sistema también puede ejecutarse como un programa independiente sin necesidad de utilizar una ventana del navegador o ejecutarse en un servidor y pueden ser sistemas con una interfaz de usuario más compleja.

Para finalizar la ejecución del sistema se hace clic en el botón “Exit” en la esquina inferior derecha.

Capítulo IV: Desarrollo de aplicaciones con Visual Prolog

En capítulos anteriores se describieron dos herramientas para el desarrollo de sistemas expertos, los cuales tenían mayormente un enfoque educativo y de fácil aprendizaje, ya que no se hacía uso de la programación. Por un lado, ES-Builder contribuye a entender la forma en que se desarrolla el árbol de decisión del sistema experto en base a los atributos, valores y conclusiones y posteriormente hacer búsquedas en este. Por otro lado, Exsys Corvid incluye otras funcionalidades que permiten desarrollar sistemas expertos más complejos con un entorno de desarrollo que facilita el diseño del sistema y de la interfaz de usuario. Por ello, este capítulo pretende describir otra herramienta para el desarrollo de sistemas expertos, de modo que sirva como complemento para las herramientas anteriores.

En este capítulo se explicará el lenguaje de programación Prolog el cual está basado en el paradigma de la programación lógica. Un paradigma representa una filosofía para diseñar soluciones y en este caso sería aplicando lenguajes de programación. La programación lógica puede entenderse entonces como un paradigma de programación basado en la lógica de primer orden, también llamada lógica de predicados, y puede verse como una deducción controlada (Martín De La Peña & Piragauta Urrea, 2016).

A diferencia de otros paradigmas de programación, la programación lógica la descripción del problema y el método para darle solución están explícitamente separadas una de la otra (Lucas, 1970). Esto se puede comprender más claramente con la ecuación propuesta por el informático teórico Robert Kowalski: ***algoritmo = lógica + control***. La programación lógica tiene diversas aplicaciones entre las que se mencionan: desarrollo de aplicaciones de inteligencia artificial, construcción de sistemas expertos, procesamiento del lenguaje natural y consultas lógicas basadas en reglas.

Además, se describirá Visual Prolog el cual es un lenguaje de programación multiparadigmas basado en el lenguaje Prolog. Este combina los paradigmas de programación lógica, funcional y orientada a objetos y cuenta con un entorno de desarrollo integrado que facilita el diseño, desarrollo, aplicación de pruebas y modificaciones de aplicaciones desarrolladas en este lenguaje.

Características del lenguaje

El lenguaje Prolog, cuyo nombre proviene del francés “PROgrammation en LOGique”, fue desarrollado a inicios de los años 70 por Alain Colmerauer y Philippe Roussel como resultado de un proyecto de procesamiento de lenguajes naturales, cuya versión definitiva apareció en 1972 y se desarrolló a partir de trabajos en demostración automática de teoremas.

Prolog permite un prototipado más rápido que con muchos lenguajes porque es más próximo a la especificación lógica del programa. Se utiliza en la computación simbólica y no-numérica y permite resolver problemas que involucran **objetos y relaciones** entre los objetos, lo cual incluye:

- Análisis de estructuras bioquímicas
- Bases de datos relacionales
- Comprensión del lenguaje natural
- Lógica matemática
- Resolución de problemas abstractos
- Resolución de ecuaciones simbólicas
- Diversas áreas de la inteligencia artificial

Otros proyectos y aplicaciones que se pueden mencionar, en los cuales se utiliza Prolog son los siguientes, de acuerdo con (Bramer, 2005):

- Sistemas de asesoramiento para aplicaciones legales
- Aplicaciones para entrenamiento
- Mantenimiento de las bases de datos del Proyecto Genoma Humano
- Análisis y medición de redes sociales
- Soporte de sistemas electrónicos para doctores
- Desarrollo de plataformas de ingeniería de software para dar soporte en el desarrollo de otros sistemas de software más complejos

El enfoque de Prolog es la descripción de hechos y relaciones acerca del problema, en lugar de la prescripción de pasos que debe tomar la computadora para resolver el problema (Clocksin & Mellish, 2012). Esto quiere decir que cuando se programa en

Prolog, este evalúa qué nuevos hechos puede inferir a partir de los hechos dados y utiliza la información de control explícita que provee el programador.

Fundamentos de Prolog

Según (Clocksin & Mellish, 2012) la programación en Prolog consiste en tres principios básicos que son:

1. Especificar algunos **hechos** acerca de los objetos y sus relaciones.
2. Definir algunas **reglas** acerca de los objetos y sus relaciones.
3. Hacer **preguntas** acerca de los objetos y sus relaciones.

En otras palabras, programar en Prolog consiste en suministrar los hechos y reglas que serán almacenados y posteriormente utilizados para responder preguntas. Los hechos y reglas se almacenan en lo que puede considerarse una base de datos del programa y estos proveen diferentes maneras de hacer inferencias entre los hechos, encontrando los valores de las variables que dan lugar a una deducción lógica.

PROgramando en LOGica

Como todo lenguaje de programación, para utilizar Prolog es imprescindible conocer la sintaxis, recordando que los elementos básicos de este lenguaje son los hechos, las reglas y las preguntas o consultas. Estos se explicarán a través de ejemplos para una mejor comprensión de la sintaxis de este lenguaje.

- **Sentencias: Hechos y reglas**

Un hecho es una proposición que puede ser cierta o falsa y es el que establece una relación entre objetos (Martín De La Peña & Piragauta Urrea, 2016). Para escribir hechos en Prolog se debe tener en cuenta lo siguiente:

- ✓ Los nombres de los objetos y relaciones deben escribirse en minúscula.
- ✓ Se escribe primero la relación y luego, entre paréntesis, los objetos separados por comas.
- ✓ Al final del hecho debe colocarse un punto “.”.

- ✓ Los nombres de los objetos dentro del paréntesis se denominan **argumentos**, mientras que el nombre de la relación se llama **predicado**. Cada uno de los argumentos por separado se denomina un **átomo**.
- ✓ El orden de los objetos del hecho es arbitrario, pero hay que ser consistente con este.
- ✓ Las relaciones pueden tener un número arbitrario de argumentos. Sea R una relación, a continuación se mencionan los tipos de relaciones más utilizadas y su sintaxis:
 - Relación unaria $\rightarrow R(a)$.
 - Relación binaria $\rightarrow R(a, b)$.
 - Relación ternaria $\rightarrow R(a, b, c)$.
- ✓ Una colección de hechos se denomina una **base de hechos** y ésta junto a las reglas se considera la base de conocimiento.

En la Figura 40 se muestra un ejemplo de la sintaxis utilizada para escribir hechos en Prolog, donde se indica que “Raquel es hija de Itza”. El orden de los argumentos es importante, ya que si en este mismo ejemplo se hubiesen colocado al revés la relación no sería la misma. También se puede observar que la relación es de tipo binaria.

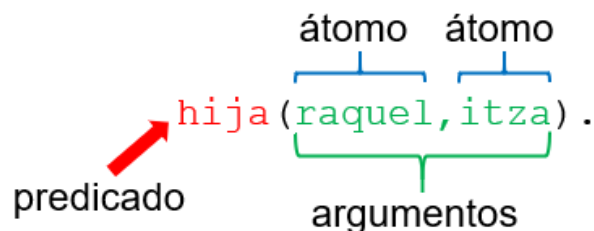


Figura 40. Ejemplo de sintaxis de un hecho en Prolog.

Una regla es una inferencia lógica que deduce un nuevo conocimiento, esto permite definir nuevas relaciones con base en otras que ya existen (Martín De La Peña & Piragauta Urrea, 2016). Las reglas no son más que una forma de representar las condiciones IF-THEN en Prolog y se utilizan cuando se quiere indicar que un hecho depende de otro grupo de hechos o para expresar definiciones. Al definir reglas en Prolog se debe considerar lo siguiente:

- ✓ Una regla consiste de dos partes: la **cabeza** y el **cuerpo**, ambos se conectan con el símbolo “:-” el cual se compone de dos puntos y un guión.
- ✓ El símbolo “:-” se conoce como **IF** o **SI**.
- ✓ Todas las reglas deben terminar con un punto “.”.
- ✓ La cabeza de la regla describe el hecho que debe ser definido por la regla, mientras que el cuerpo describe el conjunto de objetivos que deben ser satisfechos, uno después del otro, para que la cabeza de la regla sea cierta.
- ✓ El cuerpo puede componerse de uno o más objetivos y estos deben ser separados por una coma “,” la cual significa la conjunción **Y** o **AND**.

En la Figura 41 se observa un ejemplo de la sintaxis para escribir las reglas e indica que “X es descendiente de Y si Y es progenitor de X”.

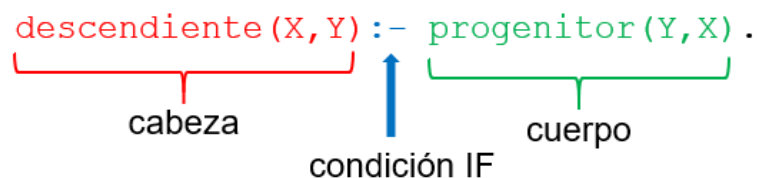


Figura 41. Ejemplo de sintaxis de una regla en Prolog.

Por otro lado, el ejemplo de la Figura 42 muestra una regla con un cuerpo que contiene dos objetivos, los cuales están separados por una coma.

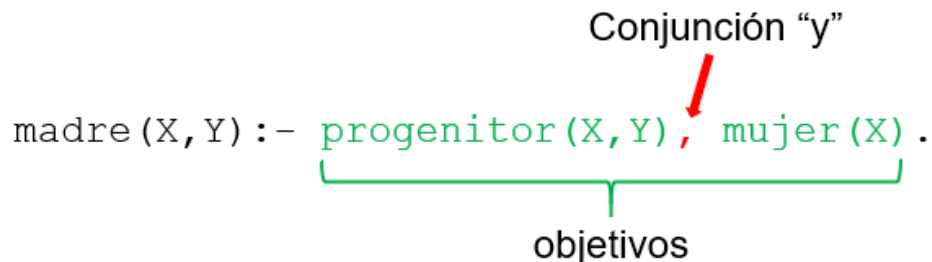


Figura 42. Ejemplo de la sintaxis de una regla con dos objetivos.

- **Consultas**

Una vez se han definido los hechos es posible hacer preguntas acerca de estos. La sintaxis para escribir una consulta o pregunta en Prolog es similar a la utilizada para escribir un hecho, con la diferencia de que se utiliza el símbolo “?-” antes de escribir el hecho que se quiere consultar.

Una vez hecha la consulta, Prolog hace una búsqueda en la base de hechos para encontrar los hechos que **unifican** o **coinciden** con el hecho en la pregunta. Dos hechos se unifican si tanto sus predicados como sus correspondientes argumentos son iguales (Clocksin & Mellish, 2012). Cuando Prolog encuentra que un hecho se unifica con la pregunta entonces responderá “**yes**” o “**true**”, de lo contrario este responderá “**no**” o “**false**”.

En la Figura 43 se muestra un ejemplo de la sintaxis para las consultas, donde se puede observar el símbolo en color rojo al principio de la consulta que le permite a Prolog diferenciar entre un hecho y una pregunta.

```
?- hija(raquel, itza).
```

Figura 43. Ejemplo de la sintaxis de una consulta en Prolog.

- **Variables: Sentencias generales**

En Prolog no sólo se pueden nombrar objetos particulares, sino que también se pueden utilizar términos conocidos como **variables** para representar objetos que no se pueden nombrar. Las variables se pueden utilizar para hacer las consultas (como se observa en la Figura 41 y la Figura 42) y las reglas, cuando se conoce el objeto al que representa la variable se dice que la variable está **instanciada**, de lo contrario se dice que es **no instanciada**.

Las variables se escriben con una letra mayúscula, una combinación de una letra mayúscula con un número o un término que empiece con letra mayúscula, dependiendo de las necesidades que se presenten durante el desarrollo del programa, por ejemplo: X, Y, X1 o Pa. Esto le facilita a Prolog distinguir entre variables y nombres de objetos particulares. Cuando se hace una consulta que contiene una variable, Prolog hace una búsqueda en la base de hechos para encontrar un objeto que la variable pueda representar.

En la Figura 44 se observa un ejemplo de una consulta utilizando una variable, donde se le pregunta a Prolog para que responda las “cosas que Raquel come”.

```
?- come(raquel, X).
```

Figura 44. Ejemplo de una consulta utilizando una variable.

De igual manera, la respuesta que Prolog dé será utilizando la variable usada en la pregunta. Siguiendo el mismo ejemplo de la Figura 44, una respuesta dada por Prolog sería de la manera mostrada en la Figura 45.

```
X = pan
```

Figura 45. Ejemplo de una respuesta dada por Prolog.

Prolog sólo dará una respuesta que indica un posible objeto al que la variable representa, sin embargo si se quiere obtener más respuestas se debe escribir un punto y coma “;” y presionar la tecla ENTER para indicarle a Prolog que continúe la búsqueda para mostrar respuesta. Cuando Prolog no encuentre otras posibles respuestas en la base de hechos entonces responderá “no”. Esto puede observarse más claramente en la Figura 46, que sigue el ejemplo de la consulta mostrada en la Figura 44.

```
X = pan;  
X = queso;  
no
```

Figura 46. Ejemplo de más de una respuesta mostrada por Prolog.

Del lenguaje natural a los programas Prolog

La sintaxis de Prolog es fácil de aprender, pero parte del aprendizaje de esta es comprender cómo se puede pasar del lenguaje natural al lenguaje Prolog. En esta sección se explicará un ejemplo que permita entender cómo llevar conocimiento del lenguaje natural al lenguaje Prolog.

El ejemplo que se utilizará consiste en un pequeño y sencillo árbol genealógico el cual se muestra en la Figura 47. Basado en este se obtendrán hechos, se definirán dos reglas que apliquen para este ejemplo, se harán consultas y se explicará de forma más clara el uso de las variables. Finalmente, se explicará cómo colocar comentarios en un programa de Prolog como parte esencial de las buenas prácticas de programación.

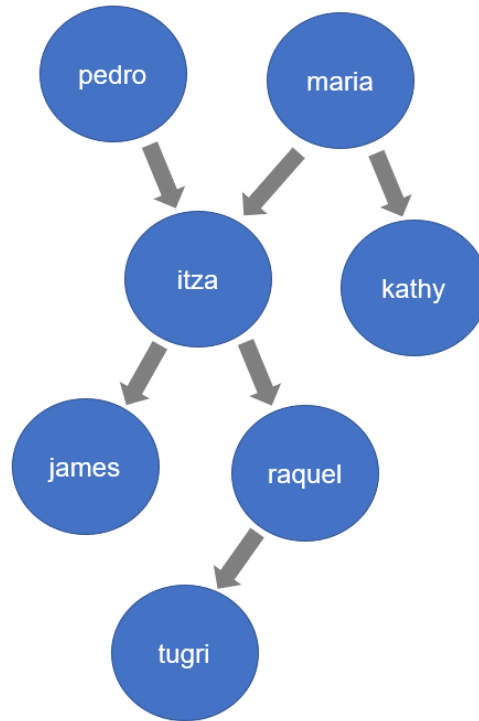


Figura 47. Diagrama que será utilizado como ejemplo para llevar de lenguaje natural a Prolog.

- **Cláusulas: Hechos y reglas**

Los hechos y reglas se denominan cláusulas y son los hechos los que le dan sentido a un programa en Prolog. Antes de escribir un hecho en Prolog se deben obtener los objetos y la relación que hay entre ellos. Por ejemplo, si se quiere decir que “Pedro es progenitor de Itza” se tendría que:

Objetos → pedro, itza

Relación → progenitor

Por lo que el hecho se escribiría como:

`progenitor(pedro, itza).`

Basado en lo anterior, entonces la base de hechos para el diagrama mostrado en la Figura 42 quedaría de la siguiente manera:

```

progenitor(pedro, itza).
progenitor(maria, itza).
progenitor(maria, kathy).
progenitor(itza, james).
progenitor(itza, raquel).
progenitor(raquel, tugri).
  
```

Con base en los hechos es posible definir las reglas. Para este ejemplo, se procederá a agregar dos reglas al programa que indiquen las relaciones “padre” y “madre”.

```
madre(X,Y):- progenitor(X,Y),mujer(X).  
padre(X,Y):- progenitor(X,Y),hombre(X).
```

La regla que define la relación de madre indica que “X es madre de Y si X es progenitor de Y y X es mujer”. Paralelamente, ocurre lo mismo con la relación padre. Como se observa, el cuerpo de una regla puede constar de más de un objetivo y es precisamente esto lo que le permitirá a Prolog hacer búsquedas en la base de hechos para responder las consultas.

Como se observa en las reglas, la relaciones “hombre” y “mujer” no existen en la base de hechos, de modo que para que las reglas tengan validez dentro del programa es necesario agregar los siguientes hechos:

```
mujer(itza).  
mujer(maria).  
mujer(kathy).  
mujer(raquel).  
hombre(pedro).  
hombre(james).  
hombre(tugri).
```

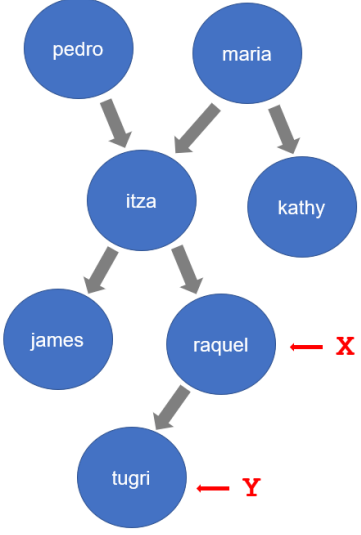
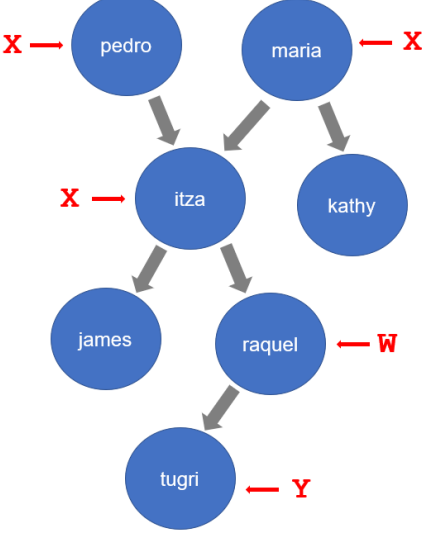
Siguiendo el mismo ejemplo, ahora se procederá a agregar una regla que indique la relación de “antepasado”. En este caso es preciso mencionar el concepto de **recursividad** el cual permite definir reglas más flexibles, como se muestra en la siguiente regla:

```
antepasado(X,Y):- progenitor(X,Y).
```

Esta regla por sí sola no permite generalizar la relación de “antecesor”, pues Prolog lo interpretaría de la siguiente manera: “X es antepasado de Y si X es progenitor de Y”. Sin embargo, sabemos que el antepasado de una persona pueden ser también los abuelos. En tal caso, se debe agregar una regla recursiva, por lo que sería:

`antepasado(X,Y):- progenitor(X,W),antepasado(W,Y).`

Para entenderlo de forma más clara, en el siguiente cuadro se hace una comparación gráfica de la interpretación que haría Prolog para cada una de las reglas, suponiendo que se quiere saber el antepasado de “tugri”:

Reglas	<code>antepasado(X,Y):- progenitor(X,Y).</code>	<code>antepasado(X,Y):-progenitor(X,Y).</code> <code>antepasado(X,Y):- progenitor(X,W),</code> <code>antepasado(W,Y).</code>
Interpretación hecha por Prolog representada gráficamente		

Los gráficos muestran la importancia de aplicar el concepto de recursividad en esta regla. Por otro lado, otra relación que puede agregarse en el programa para el ejemplo de esta sección del capítulo es la relación “hermana”. En este caso, lo más lógico es escribir la regla como “X es hermana de Y si X es mujer, W es progenitor de X y W es progenitor de Y”, de la siguiente manera:

`hermana(X,Y):- mujer(X),progenitor(W,X),progenitor(W,Y).`

Sin embargo, existe un pequeño problema: Prolog podría interpretar que X e Y pueden ser la misma persona. Para ello es conveniente utilizar un operador que indique la desigualdad entre las variables X e Y. El operador “`\=`” comprueba la desigualdad entre dos términos o variables que son distintos, entonces la regla quedaría de la siguiente forma:

```
hermana(X,Y):- mujer(X),progenitor(W,X),progenitor(W,Y),X\==Y.
```

Para una mejor comprensión de este caso, en el siguiente cuadro se hace una comparación gráfica de la interpretación que haría Prolog para cada una de las reglas, suponiendo que se quiere conocer quién es la hermana de “kathy”:

Reglas	<pre>hermana(X,Y):- mujer(X), progenitor(W,X), progenitor(W,Y).</pre>	<pre>hermana(X,Y):- mujer(X), progenitor(W,X), progenitor(W,Y),X\==Y.</pre>
Interpretación hecha por Prolog representada gráficamente		

- **Predicados (relaciones)**

Como se indicó en la sección anterior de este capítulo, las relaciones se denominan **predicados** y este corresponde a una palabra o término que permita representar la relación entre objetos.

En el siguiente hecho se tiene que “progenitor” indica la relación que existe entre los dos objetos, donde “pedro” es progenitor de “itza”.

```
progenitor(pedro,itza).
```

El predicado puede ser un sustantivo, un verbo o una frase corta que indique la relación que se quiere representar y siguiendo el ejemplo de esta sección se muestran a

continuación dos hechos que pueden agregarse a la base de hechos y en donde se observa el uso de otro predicado para indicar una nueva relación:

```
abuelo (pedro, james) .
abuela (maria, raquel) .
```

A continuación se mencionan los predicados o relaciones que hasta ahora se han utilizado para el ejemplo de esta sección y el tipo al que cada una pertenece:

Predicado o relación	Tipo de relación
progenitor madre padre antepasado abuelo abuela	Binaria
mujer hombre	Unaria

- **Variables (cláusulas generales)**

Anteriormente en este capítulo, se describió el uso de la variables en Prolog y se mencionó que estas pueden ser utilizadas para escribir las reglas y las consultas. Más adelante se explicará su uso en las consultas para una mejor comprensión de estas.

Observando las reglas del ejemplo:

```
madre (X, Y) :- progenitor (X, Y) , mujer (X) .
padre (X, Y) :- progenitor (X, Y) , hombre (X) .
```

en la regla que representa la relación de “madre” se coloca la X como la variable no instanciada que representa a la madre, mientras que Y es la variable no instanciada que representa al descendiente. Luego se observa que en el primer objetivo “progenitor” se utiliza la nuevamente la variable X porque hace referencia a la misma variable X que se utilizó en la relación “madre”.

Con esto se hace énfasis en la importancia de utilizar las mismas variables si hacen referencia a un mismo objeto, mientras que si son diferentes se puede utilizar otra letra o agregarle un número para distinguirlo, sin embargo lo recomendable es utilizar otra letra para evitar confusión. Sin embargo, no existe ningún problema si se utilizan las mismas variables en reglas diferentes, ya que Prolog las interpreta por separado.

- **Objetivos (consultas)**

Una vez se tengan las cláusulas, es posible obtener conocimiento de la base de hechos a través de las consultas o preguntas. Al igual que las reglas, las consultas se componen de objetivos que Prolog utilizará para la búsqueda en las cláusulas. Es preciso mencionar algunas características de los objetivos en Prolog:

- ✓ Se ejecutan secuencialmente de izquierda a derecha.
- ✓ Si falla uno de los objetivos, no se ejecutan los siguientes y por tanto, falla la conjunción en total.
- ✓ Si un objetivo tiene éxito, quedan unidas algunas o todas las variables.
- ✓ Si todos los objetivos tienen éxito sucede lo mismo con la conjunción.

A continuación, se muestra base de conocimiento que corresponden al programa del ejemplo presentado y en el que se basarán las consultas:

```
progenitor(pedro, itza) .  
progenitor(maria, itza) .  
progenitor(maria, kathy) .  
progenitor(itza, james) .  
progenitor(itza, raquel) .  
progenitor(raquel, tugri) .  
mujer(itza) .  
mujer(maria) .  
mujer(kathy) .
```

```

mujer(raquel).
hombre(pedro).
hombre(james).
hombre(tugri).
abuelo(pedro,james).
abuela(maria,raquel).
abuela(itza,tugri).

```

```

madre(X,Y):- progenitor(X,Y),mujer(X).
padre(X,Y):- progenitor(X,Y),hombre(X).
antepasado(X,Y):- progenitor(X,Y).
antepasado(X,Y):- progenitor(X,W),antepasado(W,Y).
hermana(X,Y):- mujer(X),progenitor(W,X),progenitor(W,Y),X\==Y.

```

Con base en lo anterior, se procederá a hacer algunas consultas. El cuadro a continuación muestra las preguntas en lenguaje natural y su equivalente en Prolog, así como la respuesta dada por este.

Pregunta en lenguaje natural		Consulta en lenguaje Prolog	Respuesta
1	¿Es itza progenitor de raquel?	?- progenitor(itza,raquel).	true
2	¿Es james progenitor de tugri?	?- progenitor(james,tugri).	false
3	¿De quién es progenitor itza?	?- progenitor(itza,X).	X = james; X = raquel; no
4	¿Quiénes son todos los X que son progenitores de Y?	?- progenitor(X,Y).	X = pedro, Y = itza; X = maria, Y = itza; X = maria, Y = kathy; X = itza, Y = james;

			X = itza, Y = raquel; X = raquel, Y = tugri; no
5	¿Quién es abuelo de james?	?- abuelo(Es_abuelo,james) .	Es_abuelo = pedro
6	¿Quién es nieto de pedro?	?- progenitor(pedro,Hijo), progenitor(Hijo,Nieto) .	Hijo = itza, Nieto = james; Hijo = itza, Nieto = raquel; no
7	¿Quién es hermana de kathy?	?- hermana(X,kathy) .	X = itza; false
8	¿Quién es el antepasado de tugri?	?- antepasado(X,tugri) .	X = raquel X = pedro X = maria X = itza; false
9	¿Quién es el hermano de raquel?	?- hermana(raquel,Hno), progenitor(Antec,raquel), progenitor(Antec,Hno) .	Antec = itza, Hno = james; false
10	¿Quién es abuelo de la mamá de tugri?	?- antepasado(Abuelo,M_Tug), progenitor(M_Tug,tugri), progenitor(X,M_Tug), progenitor(Abuelo,X), hombre(Abuelo) .	Abuelo = pedro, M_Tug = raquel, X = itza; false

Como se puede apreciar en el cuadro anterior, con una pequeña base de hechos se pueden hacer varias consultas que permitan obtener conocimiento. De igual manera, se pueden agregar más reglas para que el programa sea más flexible al momento de hacer preguntas.

- **Comentarios**

Parte de las buenas prácticas de programación es el uso de comentarios y en Prolog los comentarios pueden ayudar a que, tanto el programador como personas ajenas al desarrollo de este, comprendan el uso de las cláusulas y variables o bien, para indicar dónde comienza las cláusulas que corresponden a la base de hechos y a las reglas.

En Prolog existen dos maneras de escribir comentarios:

- ✓ Para comentarios de una sola línea se utiliza el símbolo "%". Un ejemplo de comentario utilizando una sola línea se muestra en la Figura 48.

```
madre(X,Y):- progenitor(X,Y),mujer(X). %Regla para la relacion "madre"  
padre(X,Y):- progenitor(X,Y),hombre(X). % Regla para la relacion "padre"
```

Figura 48. Ejemplo de un comentario de una sola línea.

- ✓ Para comentarios que contienen más de una línea se utiliza "/" para iniciar el comentario y "/" para indicar que finaliza. En la Figura 49 se observa un ejemplo de un comentario de más de una línea.

```
/*Base de hechos de un  
pequeño arbol genealogico*/  
progenitor(pedro,itza).  
progenitor(maria,itza).  
progenitor(maria,kathy).  
progenitor(itza,james).  
progenitor(itza,raquel).  
progenitor(raquel,tugri).
```

Figura 49. Ejemplo de un comentario de más de una línea.

Programas Visual Prolog

Como se mencionó al inicio de este capítulo, Visual Prolog es un lenguaje de programación basado en Prolog y con un enfoque basado en la combinación de los paradigmas de programación lógica, funcional y orientada a objetos. Este cuenta con un entorno de desarrollo integrado que facilita el diseño, desarrollo, aplicación de pruebas y modificaciones de aplicaciones desarrolladas en este lenguaje.

Entre las principales características de Visual Prolog se pueden mencionar:

- ✓ Basado en la programación lógica con las cláusulas de Horn, que es un sistema formal para el razonamiento acerca de objetos y sus relaciones (Mims, 2008).

- ✓ Completamente orientado a objeto.
- ✓ Utiliza patrones de emparejamiento y unificación.
- ✓ Bases de datos de hechos totalmente integradas.
- ✓ Soporta el manejo automático de memoria (Smith, 2004).
- ✓ Soporta conexiones con otros lenguajes de programación como C, C++, HTML y Java, además de incluir funciones de Windows (Smith, 2004).

Entorno de desarrollo Prolog

El entorno de desarrollo de Visual Prolog o Integrated Development Environment (IDE) de Visual Prolog está fundamentalmente diseñado para facilitar el desarrollo, pruebas y modificación de aplicaciones. Entre las principales facilidades que se pueden mencionar están:

- ✓ Representación en forma de estructura de árbol donde se muestran los módulos. Además se incluyen archivos y recursos en la ventana "Project".
- ✓ El editor de texto que permite editar y buscar declaraciones e implementaciones.
- ✓ El editor de diálogo que provee controles para el diseño de cuadros de diálogo.
- ✓ El editor de menú que permite la creación de menús.
- ✓ El editor de la barra de herramientas que permite crear diferentes tipos de barras de herramientas.
- ✓ El editor de gráficos que permite la creación, vista y edición de íconos, cursores y bitmaps.

En la Figura 50 se puede observar el IDE o entorno de desarrollo de Visual Prolog. En la derecha de la pantalla aparecen los módulos, archivos y recursos del proyecto, mientras que en la parte derecha se observa el código de este.

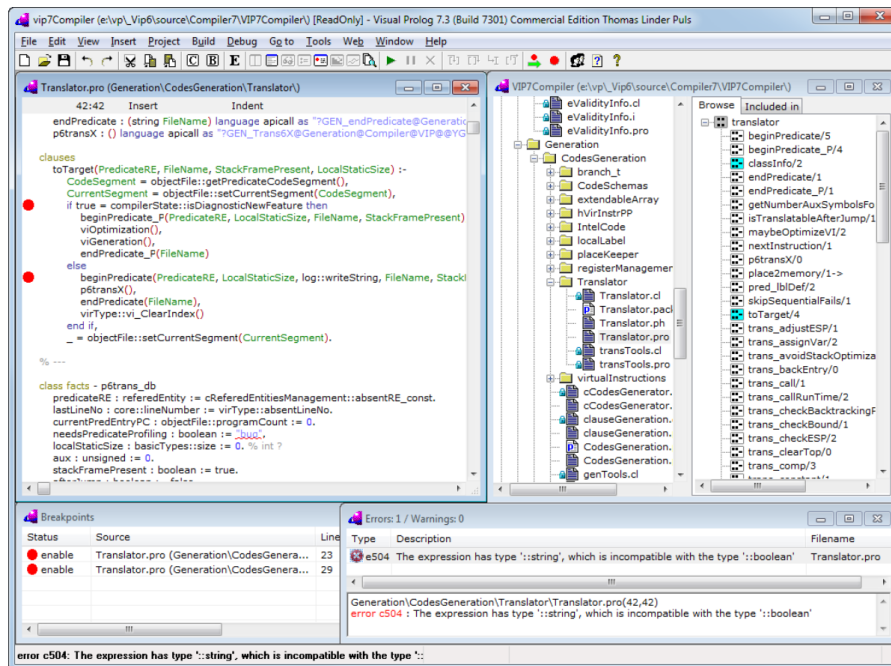


Figura 50. Entorno de desarrollo de Visual Prolog. - Tomado de Visual Prolog-Wikipedia.

El entorno de desarrollo de Visual Prolog también cuenta con los siguientes componentes:

- Un **compilador** que transforma el código fuente en código de byte. Este es sucesor del compilador de Turbo Prolog creado en en la década de los 80s y fue el primer compilador de Prolog, pero con mejoras.
- Un **intérprete** que se encarga de ejecutar el código de byte.
- Un **shell** que es una utilidad que permite probar y depurar los programas .
- Una **biblioteca de utilidades** que brindan funcionalidades para operaciones básicas como manejo de cadenas, entrada y salida de datos, etc.

Secciones básicas de un programa Visual Prolog

Parte esencial de aprender a programar en Visual Prolog es conocer las diferentes secciones que componen un programa. Un programa en Visual Prolog tiene la siguiente estructura básica:

```
domains
/* ...
domain declarations
... */
```

```

predicates
/* ...
predicate declarations
... */
clauses
/* ...
clauses (rules and facts)
... */
goal
/* ...
subgoal_1,
subgoal_2,
etc. */

```

A continuación se detalla en qué consisten cada una de estas secciones.

- **La sección “clauses”**

Esta sección se compone de cláusulas e inicia con la palabra `clauses`. Un archivo puede contener más de una de estas secciones y tiene como función especificar implementaciones de los predicados o valores iniciales de los hechos. Todas las cláusulas que corresponden a un mismo predicado o hecho deben agruparse en una sola sección de cláusulas (Boer, 2005).

Como las cláusulas permiten darles definición a los predicados, un solo predicado puede estar definido por más de una cláusula. Estas se componen de la cabeza y el cuerpo, al igual que sucede con las cláusulas en Prolog, en este caso se denominaría una regla. En el caso contrario, de que no contenga un cuerpo entonces se denominaría un hecho.

Un ejemplo de una cláusula en Visual Prolog sería el siguiente, donde se indica que “Raquel” es una persona con estatura de 160:

```

clauses
    persona("Raquel", 160) .

```

El ejemplo anterior corresponde a un hecho, pero el ejemplo, tomado de (Boer, 2005), a continuación corresponde a una regla:

```

clauses

```



```
abuelo(Nieto, Abuelo):-progenitor(Nieto,Persona),
    padre(persona, Abuelo).
```

Se puede observar que, al igual que sucede en Prolog, se utiliza el símbolo “:-” que corresponde a la sentencia IF y este separa a la cabeza del cuerpo que corresponde a las condiciones de la regla. Por otro lado, el símbolo “,” significa AND.

- **La sección “predicates”**

En esta sección se declaran los objetos del ámbito actual y un archivo puede contener más de una sección de estas. Esta sección inicia con la palabra `predicates` y para escribir predicados dentro de ella se utiliza la siguiente sintaxis:

```
predicates
    <NombrePredicado>:(<lista_de_argumentos>).
```

El nombre del predicado debe ser en minúscula, mientras que la lista de argumentos corresponde a los tipos de argumentos del objeto o predicado, los cuales pueden ser los tipos estándar de Visual Prolog o dominios que han sido definidos (esto se explicará más adelante).

Un ejemplo puede ser el siguiente, donde se indica que el predicado “persona” tiene dos valores, uno que corresponde a un entero y el otro a una cadena:

```
predicates
    persona:(integer, string).
```

También se puede especificar el nombre de la variable a la que hace referencia el tipo de argumento del objeto. En tal caso, siguiendo el ejemplo anterior, se podría declarar de la siguiente forma:

```
predicates
    persona:(integer Edad, string Nombre).
```

- **La sección “domains”**

En Prolog existen cuatro tipos de variables: `unsigned` (números no negativos), `integer` (valores enteros), `real` (valores reales), `character` y `string` (cadenas). Dentro de esta

sección del programa, la cual inicia con la palabra `domains`, los tipos de variables estándar se utilizan para crear tipos de variables propios con valores específicos. La sintaxis es la siguiente:

```
domains
    <NombreDominio>=<TipoVariable>.
```

Por ejemplo, si se tienen tres datos de una persona: peso, estatura y nombre, los dos primeros pueden ser variables de tipo `integer`, pero no pueden ser negativos. Imaginemos entonces que el máximo para la estatura de una persona es de 190 y para el peso es 200, por lo que dentro de la sección “domains” se declararían los siguientes dominios:

```
domains
    estatura=integer[0..190].
    peso=integer[0..200].
    nombre=string.
```

Como se observa en el ejemplo, esta sección permite crear un dominio con un valor delimitado, a través del uso de las variables propias de Prolog. De esta manera, los hechos pueden ser declarados de la siguiente manera:

```
facts
    persona:(nombre, estatura, peso)
```

Los dominios sólo pueden utilizarse dentro de la clase y el módulo en el que fue definido, de lo contrario es necesario hacer una referencia explícita al ámbito donde está definido y declarado el dominio (Boer, 2005).

- **La sección “goal”**

La sección “goal”, o sección de objetivos, se compone esencialmente de una lista de subobjetivos que son iguales al cuerpo de una regla (Prolog Development Center A/S, 2001). Sin embargo, se diferencian de las reglas en que un objetivo no es seguido por el símbolo “:-” y que Visual Prolog de forma automática ejecuta un objetivo cuando empieza la ejecución del programa.

Cuando Visual Prolog llama a un objetivo, trata de satisfacer el cuerpo de una regla con el objetivo. Si los subobjetivos que están en esta sección tienen éxito, entonces el programa termina de forma satisfactoria. De lo contrario, se dice que el programa falla y simplemente termina la ejecución de este (Prolog Development Center A/S, 2001).

- **La sección “database”**

Esta sección corresponde a la base de datos de hechos, también denominada sección “facts”. Los hechos describen los valores de los atributos de un objeto o clase. Los hechos se almacenan en una base de datos y los hechos pueden ser de dos tipos según la cantidad de atributos que describen (Boer, 2005):

1. **factVariable** → un solo atributo
2. **factFunctor** → dos o más atributos

Esta sección que empieza con la palabra `facts`, contiene las declaraciones de los hechos que dentro de la base de datos de hechos. Dicha base de datos puede clasificarse en dos tipos según el nivel al que pertenecen (Boer, 2005):

1. **Nivel de objeto:** Se pueden agregar, eliminar o modificar hechos a través del llamado del predicado de un objeto.
2. **Nivel de clase:** Se pueden agregar, modificar o eliminar hechos a través del llamado del predicado de una clase.

A la base de datos de hechos se le puede dar un nombre y para ello se utiliza la siguiente sintaxis, donde el identificador de la base de datos debe estar escrito en letras minúsculas:

```
facts - nombre_de_la_bd
```

Para declarar hechos que describen un solo atributo (`factVariable`), se utiliza la siguiente sintaxis donde el nombre es un identificador que inicie con una letra minúscula y el tipo de hecho es un dominio que ya se ha declarado:

```
<nombre>:<tipo_de_hecho>:=<valor_inicial>
```

A continuación se muestra un ejemplo:

```
facts
    hechoA:integer:=5
```

Para declarar hechos que describen más de un atributo (factFunctor), se utiliza la sintaxis mostrada a continuación, donde el nombre debe ser un identificador en letras minúsculas y la lista de argumentos contiene los tipos de los argumentos y opcionalmente, el nombre de cada argumento:

```
<nombre>:(<lista_de_argumentos>)
```

A continuación se muestran dos ejemplos, el primer hecho sólo indica los tipos de los argumentos, mientras que el segundo indica el tipo y el nombre de los argumentos:

```
facts
    mascota:(integer,string)
    mascota:(integer Peso, string Nombre)
```

Esta sección de un programa escrito en Visual Prolog puede considerarse la más importante, pues contiene los hechos del sistema experto y por consiguiente, son el conocimiento que le da forma al este.

- **La sección “constants”**

Se sabe que una constante es un valor que no varía durante la ejecución de un programa. En este sentido, dentro de esta sección, la cual inicia con la palabra `constants`, se definen constantes que se utilizarán en el ámbito actual y para ello se utiliza la siguiente sintaxis:

```
constants
    <NombreConstante>:<TipoConstante>=<ValorConstante>.
```

El nombre de la constante debe ser un identificador en minúscula y el tipo de constante no es más que el tipo de dato al que corresponde dicha constante. Algunos ejemplos de constantes pueden ser los siguientes:

```
constants
    numpi:real=3.14159.
    saludo:string="Bienvenido".
```

- **Secciones globales**

Visual Prolog permite la declaración de dominios, predicados y cláusulas del programa de forma global. Para ello se declaran secciones separadas de dominios, predicados y hechos globales al principio del código del programa.

Para lograr esto se requiere escribir la palabra “global” al principio de las palabras “domains”, “predicates” y “facts”, para indicar que las declaraciones que están a continuación tienen un alcance global y los nombres declarados dentro de ellos pueden ser utilizados en todos los módulos que incluyan las declaraciones de estas secciones globales. El módulo principal del proyecto que se esté desarrollando en Visual Prolog debe contener las declaraciones de todos los dominios globales y las secciones de hechos globales que están declaradas en todos los submódulos del proyecto. Además, es importante tener en cuenta que si una declaración global se modifica, todos los módulos que contengan a dicha declaración deben ser compilados nuevamente (Prolog Development Center A/S, 2001).

- **Directivas de compilación**

Las directivas de compilación en Visual Prolog pueden añadirse al programa para indicarle al compilador formas específicas en las que debe tratar al código durante el proceso de compilación (Prolog Development Center A/S, 2001).

Las directivas de compilación inician con el símbolo “#”, a continuación se mencionan las principales directivas de compilación en Visual Prolog y su función:

Directiva de compilación	Función
#include, #bininclude	Inclusión de un archivo. #include se utiliza para incluir contenidos de otro archivo dentro del código fuente del programa durante la compilación, mientras que #bininclude se utiliza para incluir durante la compilación contenidos de un archivo

	como un tipo de constante binaria dentro del código fuente del programa.
<code>#if, #then, #else, #elseif, #endif</code>	Sentencias condicionales
<code>#export, #externally</code>	Exportación e importación de clases. Se utilizan para determinar listas de clases exportadas e importadas.
<code>#message, #error, #requires, #orrequires</code>	Información sobre el tiempo de compilación.
<code>#options</code>	Opciones de compilación

Unificación y backtracking

El intérprete de Prolog utiliza dos procedimientos para hacer inferencias y deducir nueva información a partir del conocimiento expresado en la base de conocimiento: la unificación y el backtracking.

- **El mecanismo de unificación**

El mecanismo de **unificación** o **matching** es el proceso mediante el cual las variables toman valor en Prolog. A través de este es posible obtener respuestas a las consultas que se formulen y por ello la unificación constituye uno de los mecanismos esenciales de Prolog. Este consiste en buscar instancias comunes a dos átomos, uno de ellos ubicado en la cabeza de una cláusula y el otro en el cuerpo de la otra cláusula (Llorens Largo & Castel de Haro, 1993).

Para hacer coincidir o unificar los átomos de las cláusulas, Prolog seguirá las siguientes reglas también especificadas por (Llorens Largo & Castel de Haro, 1993):

- ✓ Una variable puede instanciarse con cualquier tipo de término o con otra variable. Por ejemplo, si X es una variable no instanciada e Y está instanciada a cualquier término, entonces X e Y son iguales y X quedará instanciada al valor de Y. Pero, si ninguna de las dos está instanciada, se

satisface el objetivo y se dice que las variables quedan compartidas, es decir que cuando una de ellas quede instanciada también lo hará la otra.

- ✓ Los números y átomos sólo son iguales a sí mismos, pueden también ser instanciados con una variable.
- ✓ Dos estructuras son iguales si tienen el mismo nombre y la misma cantidad de argumentos y si todos y cada uno de dichos argumentos son unificables.

Para satisfacer los objetivos a través de este mecanismo, Prolog primero hace una búsqueda desde el principio en la base de conocimiento de un hecho o la cabeza de una regla que se unifique con el objetivo que se quiere satisfacer. Luego registra en la base de conocimiento la posición e instancia todas las variables que coincidan y que hayan sido instanciadas previamente. Si encontró una regla, Prolog intentará satisfacer los subobjetivos del cuerpo de dicha regla, pero sino encuentra ningún hecho o cabeza de regla que coincida, entonces el objetivo se dice que falla e intentará resatisfacer el objetivo anterior. Es aquí donde entra en juego el proceso de backtracking.

- **Backtracking**

El proceso de **backtracking** o **reevaluación** consiste en volver a ver lo que se ha hecho e intentar resatisfacer los objetivos a través de una forma alternativa. Este proceso se lleva a cabo cuando se quiere obtener más de una respuesta a una consulta formulada, a este proceso se le denomina **generación de soluciones múltiples** (Llorens Largo & Castel de Haro, 1993).

Explicado de otra manera, Prolog utiliza este proceso para tratar de resatisfacer los objetivos. Esto trata de hacerlo en orden inverso con cada uno de los subobjetivos e intenta encontrar una cláusula alternativa que coincida con el objetivo. Cuando sucede esto, todas las variables instanciadas se dejan sin instanciar al elegir la cláusula previa y la búsqueda empieza desde donde se había dejado el registro de posición del objetivo. Prolog termina la ejecución cuando no queden ni objetivos ni registros de posición.

En otras palabras, se puede decir que el proceso de backtracking siempre regresa a la alternativa sin probar más reciente (Covington, Nute, & Vellino, 1995).

- **Predicados predefinidos**

Los predicados predefinidos en Prolog son aquellos que ya están definidos y por lo tanto, no hay necesidad de especificarlos a través de cláusulas (Escrig, Pacheco, & Toledo, 2001).

A continuación se presenta una tabla en donde se resumen los predicados predefinidos, generalmente más utilizados en Prolog con su respectivo significado, los cuales fueron resumidas por (Llorens Largo & Castel de Haro, 1993):

Uso	Predicado predefinido	Función del predicado
Adición de bases de conocimiento externas	consult (X)	Añadir las cláusulas del fichero X a la base de conocimiento
	reconsult (Y)	Las cláusulas leídas del fichero Y sustituyen a las existentes para el mismo predicado.
	halt	Salir del sistema Prolog.
Construcción de objetivos compuestos (conectivas lógicas)	X, Y	Conjunción de objetivos
	X; Y	Disyunción de objetivos
	not (X)	Se cumple si fracasa el intento de satisfacer X
Clasificación de términos	var (X)	Se cumple si X es en ese momento una variable no instanciada.
	nonvar (X)	Se cumple si X no es una variable sin instanciar en ese momento.
	atomic (X)	Se cumple si X representa en ese momento un número o un átomo.
	atom (X)	Se cumple si X representa en ese momento un átomo de Prolog.
	numeric (X)	Se cumple si X representa es en ese momento un número.

	integer (X)	Se cumple si X representa es en ese momento un número entero.
	real (X)	Se cumple si X representa es en ese momento un número real.
Control del programa	true	Objetivo que siempre se cumple.
	fail	Objetivo que siempre fracasa.
	repeat	Sirve para generar soluciones múltiples mediante el mecanismo de backtracking.
	call (X)	Se cumple si tiene éxito el intento de satisfacer X (instanciada a un término).
Operadores aritméticos y relacionales	X = Y	Se cumple si se unifican X e Y.
	X \= Y	Se cumple si fracasa X=Y.
	X < Y	Predicado menor que.
	X > Y	Predicado mayor que.
	X >= Y	Predicado mayor o igual que.
	X =< Y	Predicado menor o igual que.
	X & Y	Operador de concatenación de cadenas de caracteres.
	X + Y	Operador de suma.
	X - Y	Operador de resta.
	X * Y	Operador de multiplicación.
	X / Y	Operador de división.
	X ** Y	Operador de exponenciación.
	X ^ Y	Operador de exponenciación.
Funciones aritméticas	abs (X)	Devuelve el valor absoluto de la expresión X.
	min (X, Y)	Devuelve el menor entre X e Y.
	max (X, Y)	Devuelve el mayor entre X e Y.
	round (X)	Evalúa la expresión X y la redondea el entero más cercano.

	truncate (X)	Trunca la expresión X en un número entero.
	floor (X)	Devuelve el mayor número entero menor o igual que el resultado de la expresión X.
	ceiling (X)	Devuelve el menor número entero menor o igual que el resultado de la expresión X.
	sqrt (X)	Devuelve la raíz cuadrada de la expresión X.
	sin (X)	Devuelve el seno de la expresión X (ángulo en radianes).
	cos (X)	Devuelve el coseno de la expresión X (ángulo en radianes).
	tan (X)	Devuelve el tangente de la expresión X (ángulo en radianes).
	pi	Constante matemática <i>pi</i> (3.141593)
Manipulación de la base de conocimiento	listing (X)	Siendo X un átomo, se escriben en el fichero de salida activo todas las cláusulas que tienen como predicado dicho átomo.
	clause (X, Y)	Se hace coincidir X con la cabeza e Y con el cuerpo de una cláusula existente en la base de conocimiento.
	assert (X)	Permite añadir la nueva cláusula X a la base de conocimiento.
	asserta (X)	Permite añadir la nueva cláusula X al principio de la base de conocimiento.
	assertz (X)	Permite añadir la nueva cláusula X al final de la base de conocimiento.
	retract (X)	Permite eliminar la primera cláusula de la base de conocimiento que empareje con X.

	<code>retractall (X)</code>	Permite eliminar todas las cláusulas de la base de conocimiento que emparejen con X.
--	-----------------------------	--

- **Prolog desde un punto de vista procedural**

Prolog se diferencia de otros lenguajes de programación, como BASIC, Pascal y C, en que es declarativo. Es decir, mientras que en lenguajes de programación procedurales es necesario escribir subrutinas y funciones para indicarle a la computadora los pasos que debe seguir para resolver un problema, en el caso Prolog o de los lenguajes de programación declarativos se escriben hechos y reglas y estos le permiten a la computadora descifrar la forma de encontrar una solución.

Desde el punto de vista procedural, las reglas de Prolog son la definición de procedimientos. Es decir, las reglas son análogas a las subrutinas y funciones que se utilizan en otros lenguajes de programación. Sin embargo, la principal diferencia que existe entre una regla en Prolog y un procedimiento en otro lenguaje, es que las reglas en Prolog permiten dar múltiples definiciones alternativas al mismo procedimiento (Prolog Development Center A/S, 2001). Esto es lo que permite que cuando se ejecute un programa escrito en Prolog, se busque a través de las diferentes reglas hasta que se encuentre una que unifique.

Capítulo V: Desarrollo de aplicaciones con CLIPS

En los capítulos anteriores, se entró en detalles sobre lenguajes de programación y entornos de desarrollo que permiten modelar el conocimiento humano de una forma más intuitiva. Sin embargo, en este último capítulo se hará un mayor enfoque en la programación a través del lenguaje CLIPS, el cual es otro lenguaje que está diseñado para facilitar el desarrollo de aplicaciones para el modelado del conocimiento humano.

Introducción a CLIPS

CLIPS, que significa “C Language Integrated Production System”, es un entorno de programación creado en 1984 por Software Technology Branch (STB), NASA, en el Johnson Space Center. Este lenguaje se basa en el paradigma de programación basado en reglas, lo que lo hace útil para la creación de sistemas expertos y otros programas en los que se requiera de una solución heurística para facilitar la implementación y mantenimiento de una solución algorítmica.

Originalmente, CLIPS fue desarrollado con el propósito de brindar asistencia en la construcción de sistemas expertos aeroespaciales. Sin embargo, a partir de este acontecimiento, CLIPS se ha utilizado en una gran variedad de áreas tales como las que especifica (Riley, 1995):

- Diferentes sitios y sucursales de la NASA y la milicia
- Gobiernos, compañías y universidades
- Aplicaciones en la ingeniería de la computación como en la ingeniería del software y seguridad en redes
- Aplicaciones en medicina y genética
- Aplicaciones en botánica y agricultura

¿Qué es CLIPS?

Las primeras versiones solamente tenían capacidad para representar reglas y hechos. Sin embargo, a partir de la versión 6.0 empezó a ser posible incluir objetos en las cláusulas de las reglas. También, se pueden utilizar los objetos sin las reglas a través del envío de mensajes, de manera que deja de ser necesario el uso de la máquina de inferencia si sólo se utilizan objetos (Giarratano, 2017).

CLIPS presenta las siguientes características:

- ✓ Tiene un sistema de producción que incluye un sistema de mantenimiento de verdad con encadenamiento hacia adelante, adición dinámica de reglas y hechos y diferentes estrategias de resolución de conflictos.
- ✓ Es fácilmente integrable en aplicaciones de diferentes lenguajes y está disponible en diversas plataformas.
- ✓ Incluye COOL (Clips Object-Oriented Language) y extensiones para uso de lógica difusa o *fuzzy logic* (FuzzyCLIPS).
- ✓ Permite la integración con otros lenguajes de programación como C y Java. Además, CLIPS puede ser llamado desde un lenguaje de programación procedural y viceversa.

Por otro lado, es importante mencionar que el shell de CLIPS provee los elementos básicos de un sistema experto (Giarratano, 2017):

- ✓ Una lista de instancias y una lista de hechos que son una memoria global para datos.
- ✓ Una base de conocimiento que contiene todas las reglas.
- ✓ Un motor de inferencia que controla la ejecución de las reglas de forma completa.

Sin embargo, la característica más importante de CLIPS es que se basa en la programación dirigida por datos, en la cual los datos aquí denominados hechos y/o instancias de objetos. Estos son los que estimulan la ejecución del programa a través del motor de inferencia, el cual es el que decide cuáles son las reglas que se deben ejecutar y cuándo deben hacerlo.

En la programación en CLIPS existen tres elementos básicos y uno adicional que no es básico, pero se considera importante:

- **Tipos de datos:** Representan información y existen ocho tipos de datos primitivos:
 - ✓ **integer:** Puede ser cualquier número con o sin signo seguido por dígitos. Es equivalente al tipo de dato long integer en C.

- ✓ **float:** Son números de tipo coma flotante o decimales. Es el equivalente a la coma flotante de doble precisión en C.
- ✓ **symbol:** Secuencia de caracteres que empieza con cualquier carácter ASCII y es seguido por un 0 o cualquier otro carácter ASCII. Símbolos como los paréntesis, el símbolo “&”, la raya vertical “|”, las doble comillas y el símbolo “~” se denominan delimitadores y no deben incluirse en los símbolos. El símbolo “?” está reservado para el uso de variables.
- ✓ **string:** Es una serie de caracteres que comienzan con las doble comillas (“), seguido por un cero u otros caracteres y termina con doble comillas (”).
- ✓ **external-address:** Es la dirección de un dato externo retornado por una función que se ha escrito en otro lenguaje y ha sido integrado con CLIPS. Sólo se puede crear este tipo de dato a través del llamado a una función y se representa con el formato <Puntero-XXXXXX>, donde XXXXXX es la dirección externa.
- ✓ **fact-address:** Es la dirección o índice de un hecho y se representa con el formato <Hecho-XXX>, donde XXX es el índice del hecho.
- ✓ **instance-name:** Es el nombre de un objeto y se forma colocando un símbolo dentro de “[]”.
- ✓ **instance-address:** Es la dirección de una instancia y se representa con el formato <Instancia-XXX>, donde XXX es el nombre de la instancia.

Contienen un campo que es cualquier lugar que puede tomar un valor y se clasifican en dos tipos dependiendo del valor que pueden tomar: monocampo y multicampo.

- **Funciones:** Manipulan los datos y deben escribirse entre paréntesis para hacer llamados. Dentro del paréntesis debe ir el nombre de la función seguido por un espacio, de igual manera los argumentos deben separarse por un espacio. Los argumentos pueden ser los tipos de datos primitivos, variables o el llamado de otra función. Pueden ser de dos tipos:
 - ✓ **Definidas por el usuario:** Funciones que han sido definidas externamente del entorno de CLIPS.

- ✓ **Definidas por el sistema:** Funciones que han sido definidas internamente por el entorno de CLIPS.
- **Constructores:** Añaden conocimiento a la base de conocimiento y deben ir entre paréntesis. Estos nunca retornan un valor.
- **Comentarios:** Permiten documentar el código escrito en CLIPS y una línea de comentario inicia con el símbolo “;”.

Además, un programa básico escrito en CLIPS tiene tres componentes principales:

- **Hechos:** Representan información sobre el estado del mundo por lo que son un elemento de información fundamental. Pueden ser de dos tipos: ordenados y no ordenados.
- **Reglas:** Representan cosas que hacer con/sobre los hechos.
- **Agenda:** Conjunto de reglas.

Elementos básicos de una herramienta de un Sistema Inteligente

En el Capítulo 1 se describió a detalla una guía para la evaluación de herramientas para el desarrollo de sistemas inteligentes. Por su parte, CLIPS es una poderosa herramienta que posee diversas características que permiten el desarrollo de sistemas expertos que servirán como base para los sistemas inteligentes.

Existen tres características esenciales que convierten a CLIPS en una herramienta apropiada para desarrollar sistemas inteligentes:

1. Posee un shell basado en reglas lo que permite que la base de conocimiento, que inicialmente está vacía, se pueda llenar con la información necesaria sobre el estado inicial del problema.
2. Utiliza el encadenamiento hacia adelante, es decir, que a partir de una serie de hechos deriva otros hechos utilizando las reglas que coinciden con los hechos conocidos. Esta estrategia de inferencia posee la ventaja de proporcionar mucho conocimiento a partir de poca cantidad de información, por lo que su uso en CLIPS lo convierte en una herramienta apropiada para el desarrollo de sistemas de planificación, control, monitorización e interpretación.

3. El motor de inferencia usa internamente el denominado **algoritmo RETE** para la búsqueda de patrones, de manera que puedan encontrar reglas que coincidan con los hechos. Este es un algoritmo incremental para la búsqueda de patrones y se encarga de compilar las reglas en una red de decisión. Las entradas del algoritmo se cambian a la memoria de trabajo, mientras que las salidas se colocan en la agenda (Crowley, 2017). La Figura 51 mostrada a continuación sintetiza cómo funciona CLIPS con el uso del algoritmo RETE.

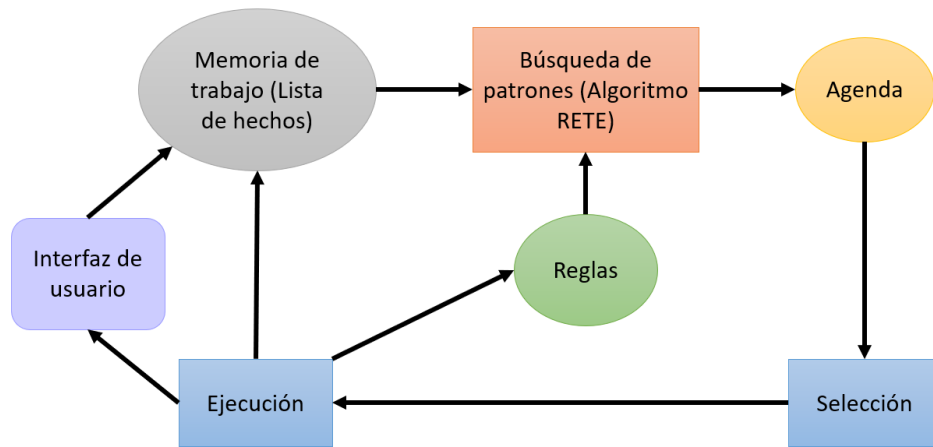


Figura 51. Funcionamiento de CLIPS con el algoritmo RETE. - Adaptado de (Crowley, 2017).

El diagrama que se muestra en la Figura 52, especifica los componentes de un sistema experto desarrollado en CLIPS. Se puede observar que los sistemas desarrollados con esta herramienta poseen la estructura básica que debe poseer todo sistema experto para formar parte de un sistema inteligente:

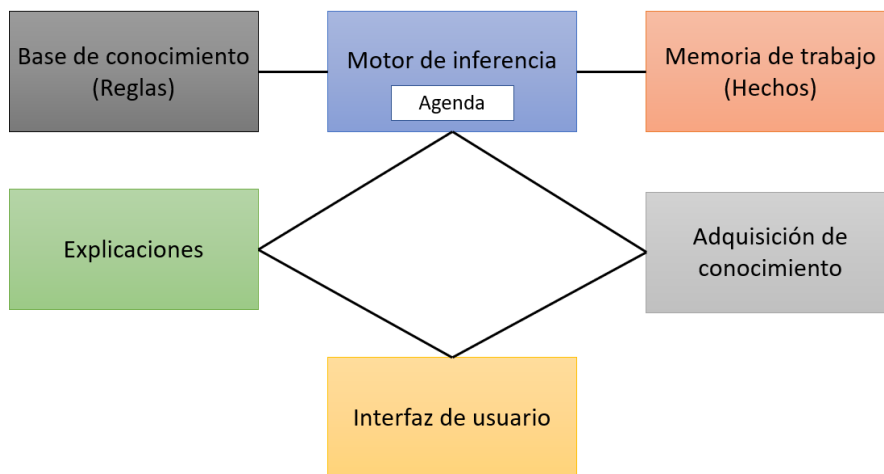


Figura 52. Componentes de un sistema experto desarrollado en CLIPS.

Del diagrama anterior se puede detallar lo siguiente:

- ✓ La memoria de trabajo se compone de la base de hechos que contiene la lista de literales positivos de lo que se conoce que es cierto acerca del universo (Pérez-Jiménez & Romero-Campero, 2004), es decir, los hechos que representan el estado inicial del problema y son los datos de los que derivan las inferencias.
- ✓ La base de conocimiento contiene una serie de reglas que pueden transformar el problema en una solución. Estas reglas son del tipo IF-THEN, en donde los hechos determinan en qué condiciones se aplican las reglas y a partir de esto, se determinan las acciones a ejecutar para dar la solución del problema, por lo que cada acción puede agregar o eliminar un hecho de la memoria de trabajo (Pérez-Jiménez & Romero-Campero, 2004).
- ✓ El motor de inferencia controla la ejecución de todo el sistema. Se encarga de comparar los hechos con las reglas para examinar cuáles reglas son aplicables y ejecutar las acciones asociadas a dichas reglas. Es importante mencionar que, como indica (Creative Commons, 2011), el motor de inferencia cuenta con una estrategia de resolución de conflictos que es la encargada de la toma de decisiones.

CLIPS tiene muchas características que facilitan el diseño y desarrollo de sistemas expertos y es esencialmente, fácil de utilizar cuando se ha utilizado con anterioridad otros lenguajes como Prolog. Su capacidad para soportar el paradigma de programación procedural lo hace útil para integrarlo con otros lenguajes de programación, lo que resultaría en sistemas inteligentes con mayor capacidad para la resolución de problemas.

Entrada y salida de CLIPS

Para comenzar a programar con CLIPS en Windows se debe hacer clic en el ícono que se muestra en la Figura 53, el cual a continuación abrirá una ventana con el cuadro "Dialog Window" o "Ventana de Diálogo", que actúa como interfaz de usuario. Dicha ventana se muestra en la Figura 54.



Figura 53. Ícono de CLIPS en Windows.

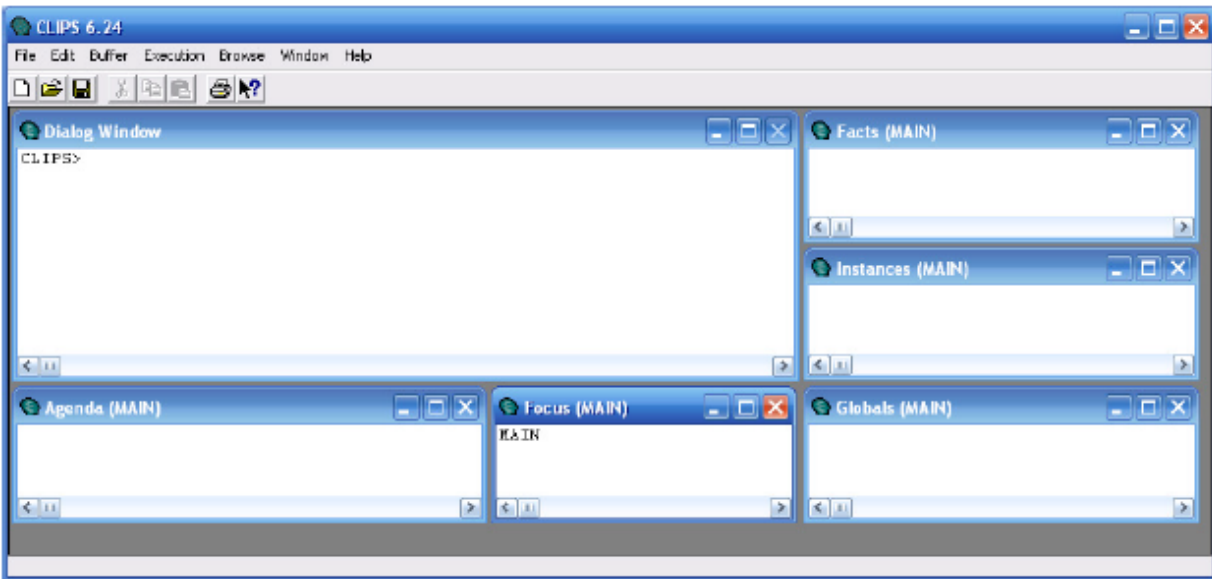


Figura 54. Ventana con cuadro de diálogo "Dialog Window" en CLIPS. – Tomado de (Cubero & Berzal, 2011).

Dentro la “Ventana de Diálogo” aparecerá la palabra `CLIPS>` que indica que se debe introducir un comando para ser evaluado y este es el método principal para interactuar con CLIPS en un entorno no integrado (Savely, Cullbert, & Riley, 2015), por lo que esto no es más que el símbolo del sistema o lo que se conoce como “command prompt”. En CLIPS esto se conoce como “top level”. A partir de este punto, se pueden escribir comandos directamente y estos deben estar entre paréntesis.

Los comandos pueden ser llamados a funciones, constructores, variables locales, variable globales o constantes. Dependiente del comando ingresado sucede lo siguiente, tal como especifica (Savely et al., 2015):

- Al hacer el llamado a una función, dicha función es evaluada y el valor retornado se imprime dentro de la Ventana de Diálogo. En este caso, el operando de la función siempre debe colocarse antes del nombre.
- La definición de un constructor crea un nuevo constructor del tipo apropiado.

- La introducción de una variable global provoca que se imprima el valor de dicha variable en la Ventana de Diálogo.
- Las variables locales pueden crearse utilizando la función `(bind)` y retenerse su valor hasta que se utilice el comando `(reset)` o `(clear)`.
- Introducir el valor de una variable local causa que se imprima el valor de dicha variable en la Ventana de Diálogo.
- Introducir una constante causa que se imprima la constante, ya que se evalúa a sí misma.

A continuación, en la Figura 55 se muestra un ejemplo de comandos que se pueden ingresar en la Ventana de Diálogo (columna izquierda) y en qué consiste cada comando y las respuestas que se imprimen en dicha ventana (columna derecha).

CLIPS> (+ 2 3)	Llamado de la función de adición para sumar los números 2 y 3.
5	Se devuelve el resultado de la suma.
CLIPS> (defglobal ?x = 2)	Definición de la variable global ?x a la que se da el valor de 2.
CLIPS> ?x	Se pregunta por el valor de la variable ?x.
2	Se devuelve el valor de la variable ?x.
CLIPS> azul	Se ingresa la constante azul.
azul	Se devuelve la constante, ya que se evalúa a sí misma.
CLIPS> (bind ?a 1)	Uso de la función <code>bind</code> para retener el valor de la variable ?a.
1	Se devuelve el valor retenido por la variable ?a.
CLIPS> (+ ?a 2)	Llamado de la función de adición para sumarle 2 al valor retenido por la variable ?a.
3	Se devuelve el valor actual de la variable ?a.
CLIPS>	En espera de que se ingrese un comando.

Figura 55. Ejemplo del uso de comandos que se pueden ingresar en la Ventana de Diálogo.

Como se mencionó anteriormente, es importante que los comandos sean escritos entre paréntesis, tal como se observa en la Figura 55. Existen otros comandos importantes que se pueden utilizar en CLIPS los cuales se especifican en la siguiente tabla.

Comando	Función
<code>(load "nombrefichero.clp")</code>	Carga un programa escrito en CLIPS a partir del fichero <code>nombrefichero.clp</code> .

(clear)	Borra todos los hechos, reglas y definiciones de CLIPS y es equivalente a cerrar CLIPS y abrirlo de nuevo.
(reset)	Pone al sistema en su estado inicial. Borra todos los hechos; coloca un hecho inicial <code>initial_fact</code> , así como todos los que el usuario defina por defecto. Debe de efectuarse antes de ejecutar cualquier programa.
(run)	Ejecuta el programa cargado en CLIPS.

Finalmente, se puede utilizar el comando `(exit)` para salir de CLIPS. Algunos de los comandos pueden ejecutarse desde el menú que se encuentra en la parte superior de la Ventana de Diálogo, como se observa en la Figura 56.

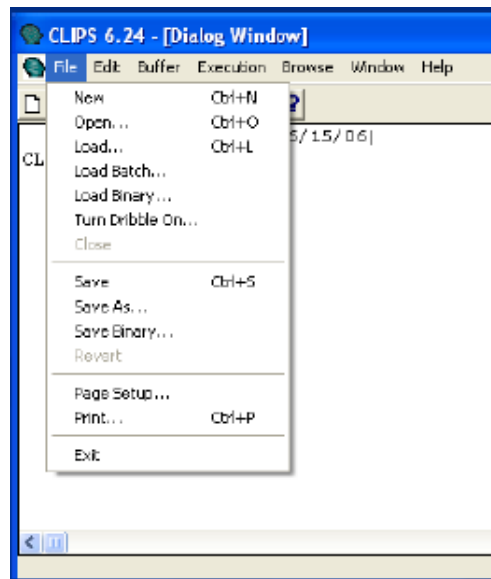


Figura 56. Menú en la parte superior de la Ventana de Diálogo desde el cual pueden utilizarse algunos comandos. – Tomado de (“CLIPS Tutorial 1,” 2009).

Elementos básicos de CLIPS

Anteriormente en este capítulo, se mencionaron los elementos básicos de un programa en CLIPS. En esta sección se hará una explicación más detallada de dichos elementos para tener una mayor comprensión de la sintaxis de CLIPS.

Hechos en CLIPS

Los hechos son los que le permiten a CLIPS conocer el estado del sistema. Estos pueden tener un solo campo o más de uno. Existen tres formas de expresar los hechos en CLIPS, para lo cual se utiliza el comando `(assert)`. A continuación se especifica la sintaxis para cada una de las tres maneras:

1. Atributo-valor: `(assert (<atributo> <valor>))`

Ejemplo:

```
CLIPS> (assert) (es-animal perro))
<Fact-0>
CLIPS> (assert) (presion-sanguinea alta))
<Fact-1>
CLIPS> (assert) (velocidad 5))
<Fact-2>
CLIPS> (assert) (velocidad 5))
FALSE
CLIPS>
```

2. Objeto-atributo-valor: `(assert (<objeto> <atributo> <valor>))`

Ejemplo:

```
CLIPS> (assert (Lassie especie perro))
<Fact-0>
CLIPS> (assert (Lassie especie perro))
<Fact-0>
CLIPS>
```

3. Relacional: `(assert (<relación> <atributo> <valor>))`

Ejemplo:

```
CLIPS> (assert (tratado-con Perez penicilina))
<Fact-2>
CLIPS> (assert (admitido-por Perez Dr-Lopez))
<Fact-3>
CLIPS>
```

Sobre los ejemplos anteriores es importante mencionar que CLIPS no sobrescribe los hechos, por lo que para modificarlos primero deben eliminarse utilizando el comando `(retract)` y luego volver a escribirlo. En el caso de que se intente introducir un hecho que ya existe, CLIPS devuelve "FALSE" indicando la imposibilidad de duplicarlo. Además

de esto, al agregar un hecho CLIPS responde con `<Fact-XX>` donde `XX` corresponde al índice numérico que le ha asignado a ese hecho.

- **Formas de ver los hechos**

Para hacer que CLIPS muestre los hechos que se han introducido con su correspondiente número de identificador se puede utilizar el comando `(facts)`. A continuación se muestra un ejemplo:

```
CLIPS> (facts)
f-0 (Lassie especie perro) CF 1.00
f-1 (Lassie domestico si) CF 1.00
f-2 (tratado-con Perez penicilina) CF 1.00
f-3 (admitido-por Perez Dr-Lopez) CF 1.00
For a total of 4 facts.
CLIPS> (reset)
CLIPS> (facts)
f-0 (initial-fact) CF 1.00
For a total of 1 fact.
CLIPS>
```

En el ejemplo mostrado se puede observar el uso del comando `(reset)` el cual permite eliminar todos los hechos que se han creado. Sucede lo mismo con el comando `(clear)`, con la diferencia de que este siempre inserta un hecho denominado **hecho inicial**.

- **Tipos de hechos: Tipos de campos**

Con los ejemplos que se han visto hasta el momento, se han estudiado los hechos denominados **hechos sin etiqueta**, que quiere decir que los campos que componen el hecho no llevan ningún tipo de etiqueta que lo identifique. Esto hace que sean sensibles al orden. Esto se observa más claramente en el siguiente ejemplo, en el que para CLIPS los dos hechos introducidos son diferentes:

Para evitar este problema existen los denominados **hechos con etiqueta o plantillas**, los cuales son insensibles al orden porque para introducir los datos se debe indicar de forma explícita el campo al que pertenecen.

El tipo al que pertenecen los campos de un determinado hecho sin etiqueta se asigna automáticamente cuando se almacena. Sucede de forma diferente con los hechos con etiqueta, ya que con estos se debe indicar el tipo de campo al que pertenece cada uno. Los tipos de campos en CLIPS pueden ser: *float*, *integer*, *symbol*, *string*, *externaladdress*, *factaddress*, *instancename* e *instanceaddress*.

- **Introducción remota de hechos**

Por otro lado, es importante mencionar que existe la posibilidad de introducir un conjunto de hechos de forma remota, lo cual se logra a través del uso del comando `(deffacts)`. Al utilizar este comando los hechos no se cargan directamente en la memoria, sino que se cargan cuando se haya reiniciado el sistema mediante el uso del comando `(reset)`.

La sintaxis para esta instrucción es la siguiente:

```
(deffacts (<hecho-1>
          (<hecho-2>
           ...
          (<hecho-n>
         )
```

A continuación se muestra un ejemplo en donde se muestra el uso de esta instrucción:

```
(deffacts estado-inicial
  (alta Jose-Martinez iop)
  (anterior Jose-Martinez infarto)
  (dolor Jose-Martinez lado-izquierdo)
  (paciente Juan-Lopez asmatico)
)
```

- **Eliminación de hechos**

Los hechos se pueden eliminar mediante el comando `(retract n)` donde `n` corresponde al índice del hecho introducido, por lo que es necesario conocer el índice del hecho que se quiere borrar. En el caso de que se quieran eliminar todos los hechos introducidos hasta el momento se utiliza el comando `(retract *)`.

Reglas en CLIPS

Las verdaderas responsables de almacenar el conocimiento del sistema experto son las reglas, ya que estas ejecutan las acciones determinadas cuando se cumplen una serie de condiciones que son las que realmente almacenan el conocimiento. Es por esto que

en un sistema experto, es imperativo tener reglas para lograr el razonamiento sobre los hechos (Carvalho, Alipio, & Neves, 2003).

En CLIPS todas las reglas son de la siguiente forma:

Si

condicion-1

condicion-2

condición-n

entonces

acción-1

acción-2

acción-n

- **Definición de reglas**

Para definir reglas en CLIPS se utiliza el constructor (defrule) y su sintaxis general es la siguiente:

```
(defrule nombre_regla "Descripción opcional entre comillas"
  (<patrón-1>           ;Miembro izquierdo de la regla
   <patrón-2>
   ...
   <patrón-n>)
=>
  (<acción-1>           ;Miembro derecho de la regla
   <acción-2>
   ...
   <acción-n>)
)                               ;Paréntesis de cierre
```

Tomando en cuenta la sintaxis, se muestran a continuación dos ejemplos de definición de reglas:

Ejemplo #1:

```
(defrule semáforo-rojo
  (luz roja)
=>
  (prinntout t "Detengase" crlf))
```


Ejemplo #2:

```
(defrule trata-infeccion "Tratamiento empírico"  
  (Perez riesgo-infeccion si)  
  (Perez infección-antes si)  
=>  
  (assert (Perez dar penicilina))  
)
```

- **Activación de una regla**

Una regla se compone de dos miembros:

1. El miembro izquierdo o **antecedente** que está formado por una serie de condiciones. Cuando todas las condiciones o hechos se cumplen, entonces la regla se activa y realiza las acciones del miembro derecho de esta. En otras palabras, son patrones que serán comparados para que unifiquen con los hechos (Kemke, 2007).
2. El miembro derecho o **consecuente** son las acciones que suelen ser afirmaciones, eliminaciones de hechos u otras acciones distintas. Dicho de otra manera, son las acciones que se ejecutarán cuando la regla se active (Kemke, 2007).

Para que un sistema basado en reglas se ejecute, este debe basarse en el cumplimiento de dichas reglas por los patrones o hechos que se introducen. Cuando no hay patrones que cumplan las condiciones necesarias, la ejecución del sistema se detiene.

- **La agenda**

La agenda es una estructura que muestra las reglas que se han activado y que después van a ser ejecutadas, colocadas en orden de ejecución. Existen dos formas de ver la agenda:

- ✓ En FuzzyCLIPS, se puede ver la ventana de agenda eligiendo la opción **Agenda Window** del menú Window.
- ✓ En la versión para terminales de texto, se utiliza el comando `(watch agenda)`, que muestra en el terminal cuando se ha activado la regla en cuestión.

Variables en CLIPS

Al igual que en otros lenguajes de programación, en CLIPS las variables se utilizan para almacenar valores, por lo que el contenido de una variable es dinámico porque puede cambiar dependiendo de los valores que se le asignen (Giarratano, 2017). El identificador o nombre de una variable debe escribirse con el símbolo “?” seguido del nombre de la variable, como se muestra en la sintaxis a continuación:

```
?<nombre>
```

Es importante asignarle un valor a la variable, de no ser así CLIPS responderá con un mensaje de error ya que no puede encontrar un valor ligado a dicha variable. Las variables globales están ligadas a todas las reglas, mientras que el resto de las variables sólo están ligadas a la regla dentro de la cual se definen. Además, es importante recordar que en CLIPS las variables no pueden ser el primer campo de un hecho.

- **Variables en el antecedente de una regla**

Normalmente las variables se usan para almacenar un valor en el antecedente de una regla, para luego utilizarlo en el consecuente de dicha regla. Dos ejemplos de esto se muestran a continuación:

Ejemplo #1:

```
(defrule colorcoche
  (coche ?color)
=>
  (assert (coche ?color))
```

Ejemplo #2:

```
(defrule quienesquien
  (cazador ?cazador ?cazado)
=>
  (printout t ?cazador "dispara al" ?cazado crlf))
```

- **Variables que almacenan direcciones**

Una variable puede almacenar la dirección de un hecho y es útil cuando se quiere conocer la posición que ocupa en la memoria un determinado hecho que se ha

introducido y se quiere eliminar. Para ello se asigna la dirección a una variable utilizando el operador “<-”. A continuación se muestra un ejemplo donde se utiliza una variable para almacenar una dirección:

```
(defrule quienesquien2
  (?asesinato <- (cazador ?cazador ?cazado)
=>
  (printout t "Hecho numero" ?asesinato ":" crlf ?cazador "ha
matado a" ?cazado crlf)
```

- **Comodines: Monocampo y multicampo**

Algunas veces sólo se especifica una parte de un hecho y el resto de este puede contener cualquier valor o simplemente estar vacío. Para evitar tener que asignar el valor de estos campos a una variable, se puede usar un comodín para detectar la presencia de un campo no vacío. Un ejemplo de esto se muestra a continuación:

```
CLIPS> (defrule agencia-matrimonial)
(nombre Pedro ?)
=>
(assert (encontrado-Pedro-apellido si))
CLIPS> (assert (Pedro Gonzalez Lopez))
CLIPS> fact-0
CLIPS> (assert (Pedro))
CLIPS> fact-1
CLIPS> (assert (Pedro Lopez))
CLIPS> fact-2
CLIPS>
```

Como se observa en la segunda línea del ejemplo, se utiliza el símbolo “?” que podría denominarse **comodín monocampo**, ya que sustituye exactamente a un campo sencillo en una afirmación particular. Además, se puede observar que sólo el segundo hecho es el que activa la regla porque es el único que posee el símbolo “Pedro”. Esto se denomina **variable monocampo**.

En cambio, para el uso de un **comodín multicampo** se usa el símbolo “\$?”, ya que equivale a cero o más campos en un hecho. Tomando en cuenta el ejemplo anterior, si se cambia el símbolo de los comodines entonces los tres hechos activarían la regla. En este caso se dice que es una **variable multicampo**.

La cola de activaciones

CLIPS hace uso de una estructura de cola para el almacenamiento y operación con las reglas que se han activado en un momento dado. El mecanismo de activación de las reglas que utiliza CLIPS consiste en lo siguiente: CLIPS realiza un ciclo con todas las reglas en el orden en que éstas han sido introducidas, comprobando qué reglas cumplen las condiciones necesarias para ser activadas, es decir, que se cumplan todos y cada uno de los patrones que hay en su antecedente. Cuando esto sucede, la coloca en cola, de modo que la última regla que se ha activado sería la última en ejecutarse.

Además de esta estructura, CLIPS utiliza para almacenar las reglas un **mecanismo de prioridades manipulable por el usuario**. Esto quiere decir que si dos reglas se activan simultáneamente, el orden en el que se colocarán en la cola es tal que la de mayor prioridad se colocará primero.

Hechos más complejos

Anteriormente, se mencionó que uno de los tipos de hechos, los denominados **hechos con etiqueta** o **plantilla**, son insensibles al orden. Estos consisten en una serie de campos que almacenan datos relativos a un determinado hecho. Dichos campos están etiquetados, por lo que no es necesario introducir valores en un orden predeterminado. Además, es posible introducir un valor por defecto para los campos, de forma que el usuario no tiene la necesidad de introducir todos y cada uno de los valores de los campos al afirmar un hecho perteneciente a un tipo establecido.

En otras palabras, las plantillas pueden considerarse contenedores de múltiples hechos, ya que cada hecho ocupa un espacio en la plantilla (Watkin, 2017).

- **Definición de plantillas**

Las plantillas se pueden definir usando la instrucción `(deftemplate)`, cuya sintaxis se indica a continuación:

```
(deftemplate <nombre de plantilla> "Comentario opcional entre comillas"
```

```
    <definición campo-1>
```

```
    <definición campo-2>
```

```
    ...
```

<definición campo-n>

)

<definición campo> = **(Slot** <nombre campo> [(**type** <tipo al que pertenece>)

[**default** <valor por defecto>]] |

Multislot <nombre campo_múltiple> [(**type** <tipo al que pertenece>) [**default** <valor por defecto>]]...)

Para verlo de manera más clara a continuación se muestra un ejemplo donde se utiliza la instrucción `(deftemplate)` para definir la plantilla “paciente” y luego se introduce un hecho de tipo paciente con sus distintos campos:

```
(deftemplate paciente
  (slot nombre)
  (multislot apellidos)
  (slot edad)
  (slot sexo)
  (slot volumen_pulmonar)
)
(assert (paciente
  (nombreapellidos Jose Quintero Garcia)
  (edad 52)
  (sexo varon)
  (volumen_pulmonar 20))
)
```

Cuando se declara una plantilla se puede incluir el uso de los tipos de datos primitivos para describir el tipo de campo que se está definiendo. Para ello se puede utilizar:

- La primitiva `(type <tipo-campo>)` para indicar el tipo de dato en el campo.
- La primitiva `(default <valor-por-defecto>)` para indicar el valor por defecto, que es el valor que toma el campo cuando no se introduce o cuando se hace `(reset)`.
- La primitiva `(default ?DERIVE)` selecciona el valor apropiado para cada tipo de campo.
- La primitiva `(range <valor-inicial> <valor-final>)` se usa para definir el rango de valores permitidos en el campo.
- Otras primitivas como: `(allowed-symbols)`, `(allowed-strings)`, `(allowed-numbers)`, `(allowed-integers)`, `(allowed-floats)` y

(allowed-values), se pueden utilizar para enumerar los valores permitidos para un campo en concreto.

A continuación, se muestra el ejemplo anterior pero esta vez modificado, haciendo uso de las primitiva y los valores por defecto:

```
(deftemplate paciente
  (slot nombre
    (type STRING)
    (default ?DERIVE))
  (multislot apellidos
    (type SYMBOL)
    (default ?DERIVE))
  (slot edad
    (type INTEGER)
    (default 30))
  (slot sexo)
    (type SYMBOL)
    (allowed-symbols V v M m)
  (slot volumen_pulmonar
    (type REAL)
    (default 2.5))
  (slot estado
    (type SYMBOL)
    (default baja)
    (allowed-symbols alta baja))
)
```

- **Uso de plantillas en reglas**

Las plantillas se pueden utilizar en reglas de la misma manera que los hechos no estructurados. El siguiente ejemplo cómo se puede manipular una plantilla en una regla:

```
(defrule BuscoaJacques
  (paciente (nombre Jacques) (apellidos $?apellidos))
=>
  (printout t "Te he encontrado, Santiago" ?apellidos crlf)
)
(defrule adios
  ?sano <- (paciente (nombre ?nombre) (estado alta))
=>
  (printout t "El paciente " ?nombre "ha sido dado de alta" crlf)
  (retract ?sano)
)
```

Finalmente, se puede utilizar la instrucción `(modify)` para modificar los campos de una plantilla.

Otros aspectos de control

Los aspectos de control en CLIPS ayudan en la construcción de reglas y esta sección del capítulo se explicarán los más importantes:

- **Restricción ~**

Actúa negando el valor sobre el que actúa. Para comprender mejor su uso, se puede observar el siguiente ejemplo, en el que se quiere escribir una regla que niegue el paso a toda persona de la base de conocimiento que no tenga como nombre "Pedro":

```
(defrule prohibirpaso
  (persona (nombre ~Pedro))
=>
  (printout t "Prohibido el paso" crlf)
  (assert (pasar no))
)
```

Para el ejemplo mostrado, la regla se activará para todos aquellos hechos del tipo de persona cuyo nombre no sea "Pedro". Además, dará un mensaje prohibiendo el paso y produciendo el hecho `(pasar no)`.

- **Restricción |**

Se utiliza para combinar varios hechos. Por ejemplo, como se muestra a continuación, supongamos que se quiere definir una regla que permita el paso a "Pedro" y "Lola":

```
(defrule permitirpaso
  (persona (nombre Pedro|Lola))
=>
  (printout t "Puede pasar" crlf)
  (assert (pasar si))
)
```

- **Restricción &**

Se utiliza para conectar varias restricciones en unión. Un ejemplo de esto se observa a continuación:

```
(defrule permitirpaso2
  (persona (nombre ?name&Pedro|Lola))
=>
  (printout t "Puede pasar" ?name crlf)
  (assert (pasar si))
)
```

- **Operaciones aritméticas en CLIPS**

CLIPS proporciona las funciones y operaciones básicas para realizar operaciones aritméticas sobre datos de tipo numérico: +, -, *, /, div, max, min, abs, float e integer. Su significado es análogo al que tienen en otros lenguajes de programación.

Las expresiones en CLIPS se caracterizan por utilizar la **notación prefija**, que consiste en anteponer el operador a los operandos de la función dada. Por ejemplo, si se quiere representar la suma de dos se debe escribir de la siguiente manera:

```
(+ 3 4)
```

La función anterior representa la suma entre los número 3 y 4 y se observa el signo de suma antes de los operandos. Es importante que sea escrito entre paréntesis.

Las funciones pueden utilizarse en el antecedente y consecuente de una regla, como se muestra en el ejemplo a continuación donde se quiere calcular el resultado de la suma de dos números:

```
(defrule Suma
  (números ?x ?y)
=>
  (assert (resultado_suma (+ ?x ?y)))
)
```

Para que una función sea evaluada en el antecedente de una regla se debe utilizar el signo "=", que le informa a CLIPS de que se debe evaluar la expresión que la sigue en lugar de usarla como una comprobación de patrones o hechos. A continuación se presenta un ejemplo:

```
(defrule Hipotenusa
  (números ?x ?y)
  (stock ?ID =(sqrt(+(** ?x 2) (** ?y 2))))
=>
  (printout t "Stock ID =" ?ID crlf)
```


)

- **Función de asignación Bind y Create\$**

La asignación de un valor a una variables en el antecedente de una regla mediante correspondencia de patrones es análogo a enlazar un valor a una variable en el consecuente usando la función `(bind)`. El ejemplo que se muestra a continuación es una nueva versión del ejemplo anterior, donde se suman dos números, pero esta vez utilizando la función `(bind)`:

```
(defrule Suma2
  (números ?x ?y)
=>
  (assert (respuesta (+ ?x ?y)))
  (bind ?respuesta (+ ?x ?y))
  (printout t "La respuesta es " ?respuesta crlf)
)
```

La función `(bind)` puede utilizarse para enlazar valores monocampo y multicampo a una variable. Dicha función se usa para enlazar cero, uno o más valores a una variable sin necesidad de utilizar el operador "\$", ya que al utilizar `(bind)` en el antecedente los argumentos indican a CLIPS exactamente cuántos valores enlazar.

Para utilizar `(bind)` para la creación de un valor multicampo, se puede utilizar en conjunto con la función `(create$)`. Su sintaxis es la siguiente:

```
(create$ <arg1><arg2>...<argN>)
```

Como se observa, a la función se le pueden agregar cualquier número de argumentos para crear un valor multicampo. A continuación se muestra un ejemplo donde se utiliza la función `(create$)`:

```
(defrule demostracion_bind
  (initial_fact)
=>
  (bind ?paciente1 (create$ Pedro Sanchez))
  (bind ?paciente2 (create$ Juan Jimenez))
  (bind ?paciente3 (create$ Jose Perez))
  (printout t "El paciente 1 es " ?paciente1 crlf
            "El paciente 2 es " ?paciente2 crlf)
```

```
        "El paciente 3 es " ?paciente3 crlf "fin" crlf)
)
```

- **Read y Readline**

La función `(read)` se utiliza para leer información introducida desde el teclado o desde un fichero abierto previamente. A continuación se muestra un ejemplo utilizando esta función:

```
(defrule lee-entrada
  (initial-fact)
=>
  (printout t "Introduzca un color primario" crlf)
  (assert (color (read))))
)
```

Sin embargo, esta función presenta ciertas limitaciones:

1. Sólo lee un campo. Si se quiere introducir varias palabras separadas por espacio, estas deben ser encerradas entre comillas para que CLIPS interprete la cadena como un solo hecho.
2. No pueden introducirse paréntesis, excepto que estos vayan encerrados entre comillas. Por lo tanto, no se puede afirmar un hecho que contiene paréntesis.

La función `(readline)` se utiliza para leer valores múltiples hasta que se finalice la introducción de un retorno de carro. Esta función permite leer los datos como una cadena, por lo que para afirmar hechos introducidos mediante `(readline)`, se utiliza la función `(assert-string)` que convierte una cadena de caracteres en un hecho de tipo no cadena. El siguiente ejemplo muestra el uso de las funciones `(readline)` y `(assert-string)`:

```
(defrule test-readline
  (initial-fact)
=>
  (printout "Introduzca una cadena" crlf)
```

```
(bind ?cadena(readline))
(assert-string(str-cat "(" ?cadena ")"))
)
```

Como la función `(assert-string)` requiere utilizar paréntesis en el argumento que se va a convertir en un hecho, se utilizó también la función `(str-cat)`, que permite concatenar una serie de subcadenas produciendo una sola.

- **Otras características**

CLIPS tiene un conjunto de funciones lógicas y aritméticas predefinidas que pueden ser utilizadas para la comparación de números, variables y cadenas en el antecedente de una regla. La siguiente tabla resume estas funciones predefinidas:

Lógicas		Aritméticas		Comparación	
not	No booleano	/	División	eq	igual
and	Y booleano	*	Producto	neq	no igual
or	O booleano	+	Suma	=	igual
		-	Resta		

Además de las restricciones descritas anteriormente, también existe lo que se denomina **restricción predicado** que se utiliza para evaluar condiciones con campos más complejos. Este tiene el propósito de admitir o rechazar un campo dependiendo del valor de una expresión booleana y es muy útil con patrones de tipo numérico. La restricción no se satisfará y la condición no se cumplirá si el resultado de la expresión es FALSE.

La función predicado puede devolver un valor FALSE o no-FALSE. Una restricción predicado se constituye cuando hay dos puntos ":" seguido de una función predicado. Los dos puntos pueden ir precedido de "&", "|" o de "~". Generalmente se utiliza en combinación con la restricción conectiva "&:". La siguiente tabla resume las funciones predicado que posee CLIPS:

Función predicado	Evalúa si <arg> es
<code>(evenp <arg>)</code>	Número par

<code>(floatp <arg>)</code>	Número en punto flotante
<code>(integerp <arg>)</code>	Número entero
<code>(lexemep <arg>)</code>	Símbolo o cadena
<code>(numberp <arg>)</code>	Float o integer
<code>(oddp <arg>)</code>	Número impar
<code>(pointerp <arg>)</code>	Dirección externa
<code>(sequencep <arg>)</code>	Valor multicampo
<code>(stringp <arg>)</code>	Cadena
<code>(symbolp <arg>)</code>	Símbolo

Finalmente, el constructor `defglobal` permite definir un valor o una variable como global, lo cual es bastante útil cuando nos interesa que se tengan valores que sean conocidos globalmente por el sistema experto.

Bibliografía

- Abraham, A. (2004). *Intelligent Systems: Architectures and Perspectives*, 1–35.
https://doi.org/10.1007/978-3-7908-1770-6_1
- Bielawski, L., & Lewand, R. (1991). *Integrating Expert Systems Design*.
- Boer, T. W. De. (2005). A Beginners ' Guide to Visual Prolog. *Cataloging*, 41(1), 281. https://doi.org/10.1300/J104v41n01_07
- Bramer, M. (2005). *Logic Programming with Prolog*. <https://doi.org/10.1007/1-84628-212-8>
- Carvalho, P., Alipio, P., & Neves, J. (2003). Using CLIPS to Detect Network Intrusions. *Lecture Notes in Computer Science*, 2902, 341–354. Retrieved from
http://en.scientificcommons.org/43236621%5Cnpapers2://publication/doi/10.1007/978-3-540-24580-3_40
- Clocksinn, W. F., & Mellish, C. S. (2012). *Programming in Prolog: Using the ISO Standard*. Springer Berlin Heidelberg. Retrieved from
<https://books.google.es/books?id=wuERBwAAQBAJ>
- Covington, M. a, Nute, D., & Vellino, A. (1995). *Prolog programming in depth*.
- Creative Commons. (2011). CLIPS - Code Snippets.
- Crowley, J. L. (2017). *Intelligent Systems : Reasoning and Recognition*.
- Cubero, J. C., & Berzal, F. (2011). *Sistemas Inteligentes de Gestión*.
- Dahiya, P. K., Chaudhary, P., Saini, J. S., & Kumar, S. (2015). *Intelligent Systems : Features , Challenges , Techniques , Applications & Future Scopes*, (December 2007).
- Edward A Feigenbaum. (1982). *Knowledge Engineering in the 1980s. Dept of Computer Science Stanford University Stanford CA USA*.

Escrig, M. T., Pacheco, J., & Toledo, F. (2001). *El Lenguaje de Programación PROLOG*.

Exsys Corvid. (2007). Exsys Corvid Advanced Tutorial.

Exsys Corvid. (2011). Corvid - Quick Start Guide.

Exsys Corvid. (2015). New Academic and Research Package Options for Exsys Corvid Software Tools. <https://doi.org/10.15713/ins.mmj.3>

Giarratano, J. C. (2017). CLIPS 6.4 User's Guide.

Hernández López, J. (2008). *Diseño de Sistemas Inteligentes Híbridos VLSI con base en Técnicas Difuso-Evolutivas*.

Kemke, C. (2007). Expert Systems Components of CLIPS. *Expert Systems*.

Llorens Largo, F., & Castel de Haro, M. J. (1993). Prácticas de Lógica. Prolog.

Lucas, P. (1970). Introduction to PROLOG.

Martín De La Peña, L. E., & Piragauta Urrea, A. A. (2016). Programación Lógica.

Mims, J. (2008). Visual Prolog Tutorial. <https://doi.org/10.3929/ethz-a-000596262>

Osorio, F. S., & Vieira, R. (1999). Sistemas Híbridos Inteligentes. *XIX Congresso Da SBC ENIA*.

Pérez-Jiménez, M., & Romero-Campero, F. (2004). A CLIPS simulator for recognizer P systems with active membranes. *Second Brainstorming Week on Membrane Computing*, 387–413. Retrieved from <http://www.gcn.us.es/2BWMC/bravolpdf/CLIPS.pdf>

Prolog Development Center A/S. (2001). Visual Prolog Language Tutorial.

Riley, G. (1995). CLIPS: An Expert System Building Tool.

Rodríguez R., A., Hernández C., J., & Plácido C., A. (2006). Herramientas de

construccion de sistemas expertos. *Ingeniería Del Conocimiento*, 3.

Savely, R., Cullbert, C., & Riley, G. (2015). CLIPS Reference Manual Basic Programming Guide.

Smith, T. (2004). Prolog: Beyond the text & Summary Artificial Intelligence Programming in Prolog.

Watkin, J. L. (2017). An introduction to the Python programming language. *Emerging Languages*, 1–4.

Anexos 1: Pruebas Rápidas

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Herramientas aplicadas a la Inteligencia Artificial

Parcial #1

Nombre: _____ Cédula: _____ Grupo: _____
Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: ____/____

I Parte: Cierto y Falso

1. _____ Con el desarrollo de la inteligencia artificial (IA) y la evolución tecnológica, los sistemas inteligentes deben ser capaces de responder ante las exigencias de los usuarios a través de inferencias.
2. _____ El ciclo de vida del sistema experto se compone solamente de 3 fases.
3. _____ Algunas de las fases del desarrollo del sistema son: implementación, análisis y diseño de sistemas, prototipado rápido entre otras.
4. _____ Los sistemas construidos con métodos útiles sin ningún conocimiento específico del dominio de aplicación se conocen como shells (conchas), sistemas esqueléticos o simplemente herramientas de IA.

II Parte: Desarrollo

1. Haga una comparación sobre el sistema clásico y el sistema experto

BUENA SUERTE

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Herramientas aplicadas a la Inteligencia Artificial

Parcial #2

Nombre: _____ Cédula: _____ Grupo: _____
Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: ____/____

I Parte: Llene los espacios en blanco

1. Escriba las cuatro técnicas principales en la utilización de Soft – Computing:

_____, _____,
_____ y _____.

2. Escriba 2 aplicaciones de las diversas áreas que tiene la Inteligencia Artificial:

_____ y _____.

3. Son 2 capacidades del sistema expertos: _____ y

_____.

4. Es una limitación del sistema experto: _____

II Parte: Desarrollo

1. Explique los tres componentes de los sistemas expertos

BUENA SUERTE

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Herramientas aplicadas a la Inteligencia Artificial

Parcial #3

Nombre: _____ Cédula: _____ Grupo: _____
Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: ____/____

I Parte: Desarrollo

1. Describa los cinco elementos más importantes de la estructura de la evaluación
2. Mencione los pasos para evaluar herramientas para la creación de sistemas inteligentes
3. Explique con sus palabras que es un Expert System Builder
4. Mencione 2 características del ES- Builder
5. Enumere las etapas del desarrollo del Exppert System Builder

BUENA SUERTE

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Herramientas aplicadas a la Inteligencia Artificial

Parcial #4

Nombre: _____ Cédula: _____ Grupo: _____
Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: ____/____

I Parte: Cierto y Falso

1. _____ Exsys Corvid es una poderosa herramienta de propósito general ampliamente utilizada para desarrollar aplicaciones interactivas de sistemas expertos.
2. _____ Exsys Corvid provee múltiples formas de describir la lógica a través del uso de reglas IF- THEN basadas mas que todos en variables.
3. _____ Exsys Corvid permite que una tarea simple no se requiera conocimientos de programación en HTML para el uso del sistema experto en línea.
4. _____ Para que un sistema experto funcione de manera eficiente necesita que se integre con otros recursos para obtener datos, guardar resultados o monitorear procesos.
5. _____ Exsys Corvid es una herramienta que permite construir sistemas de manera rápida, gracias a la simplicidad de la sintaxis de las reglas, permitiendo que el experto en dominio construya el sistema por si mismo.

BUENA SUERTE

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Herramientas aplicadas a la Inteligencia Artificial

Parcial #5

Nombre: _____ Cédula: _____ Grupo: _____
Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: ____/____

I Parte: Llene los espacios

1. Mencione dos áreas de aplicación que se construyen con Corvid:
_____ y _____.
2. Mencione dos áreas de aplicación de la herramienta Exsys Corvid:
_____ y _____.
3. Si nos hacen la siguiente pregunta: “¿Puede el experto explicar a otra persona cómo resolver el problema”? que sucede si la respuesta es “NO”:
_____.
4. En el área de formato de variables cuales son las tres pestañas que comúnmente se utilizan: _____, _____ y _____.

II Parte: Desarrollo

1. Describa los pasos para la construcción de una aplicación con Exsys Corvid

BUENA SUERTE

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Herramientas aplicadas a la Inteligencia Artificial

Parcial #6

Nombre: _____ Cédula: _____ Grupo: _____
Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: ____/____

I Parte: Cierto y Falso

1. _____ Las variables de tipo lista estática son las que tienen una lista específica de múltiples valores y puede haber cualquier cantidad de valores en la lista.
2. _____ Las variables numéricas son las que se les asignan un valor numérico y puede obtenerse preguntándole al usuario.
3. _____ Corvid tiene solamente dos tipos de variables que se utilizan para construir expresiones booleanas.
4. _____ Las condiciones IF se muestran como un grupo relacionado de nodos que proveen condiciones para una misma variable.
5. _____ El bloque de comandos o Command Block en Corvid se encarga de decirle al sistema qué hacer, a diferencia de los bloques lógicos que indican cómo hacer las cosas.

BUENA SUERTE

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Herramientas aplicadas a la Inteligencia Artificial

Parcial #8

Nombre: _____ Cédula: _____ Grupo: _____
Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: ____/____

I Parte: Llene los espacios

1. El lenguaje de programación Prolog esta basado en el paradigma de programación:

_____.

2. Escriba dos objetos y relaciones que permite el prototipado Prolog:

_____ y _____.

3. Mencione un principio básico de Prolog:

4. En Prolog, no solo se pueden nombrar objetos particulares, sino también se pueden utilizar términos conocidos como: _____.

5. Escriba una facilidad del entorno de desarrollo Prolog:

_____.

BUENA SUERTE

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Herramientas aplicadas a la Inteligencia Artificial

Parcial #9

Nombre: _____ Cédula: _____ Grupo: _____
Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: ____/____

I Parte : Cierto y Falso

1. _____ Las primeras versiones de CLIPS solamente tenían capacidad para representar reglas y hechos.
2. _____ La característica más importante de CLIPS es que se basa en la programación dirigida por datos.
3. _____ Symbol es una serie de caracteres que empieza con las dobles comillas
4. _____ Float son números de tipo coma flotante o decimales.
5. _____ External- Address es la dirección o índice de un hecho y se representa con el formato <Hecho-XXX>.

II Parte: Desarrollo

1. Explique qué significa CLIPS

BUENA SUERTE

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Herramientas aplicadas a la Inteligencia Artificial

Parcial #10

Nombre: _____ Cédula: _____ Grupo: _____

Profesor: Dr. Carlos A. Rovetto

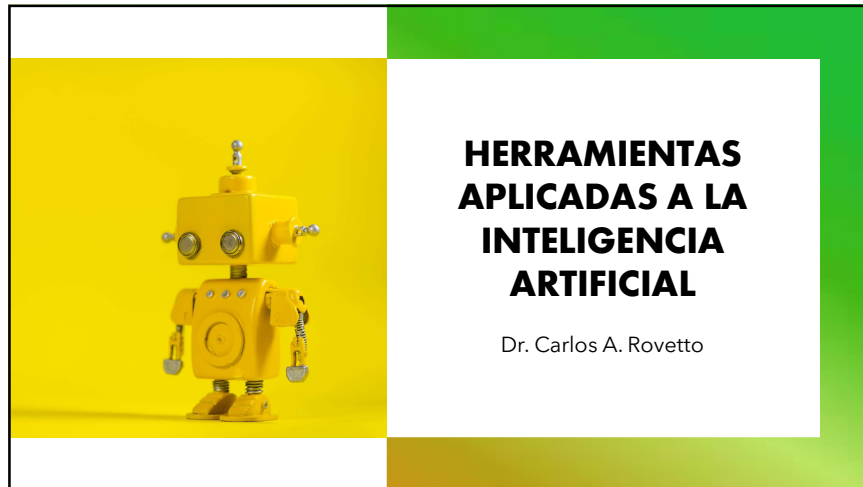
Puntos Obtenidos: ____/____

I Parte: Desarrollo

1. De un ejemplo sobre una variable que almacena dirección
2. De un ejemplo sobre una variable en el antecedente de una regla
3. De un ejemplo sobre una Comodines: Monocampo y Multicampo
4. De un ejemplo sobre restricción &
5. De un ejemplo sobre Read y Readline.

BUENA SUERTE

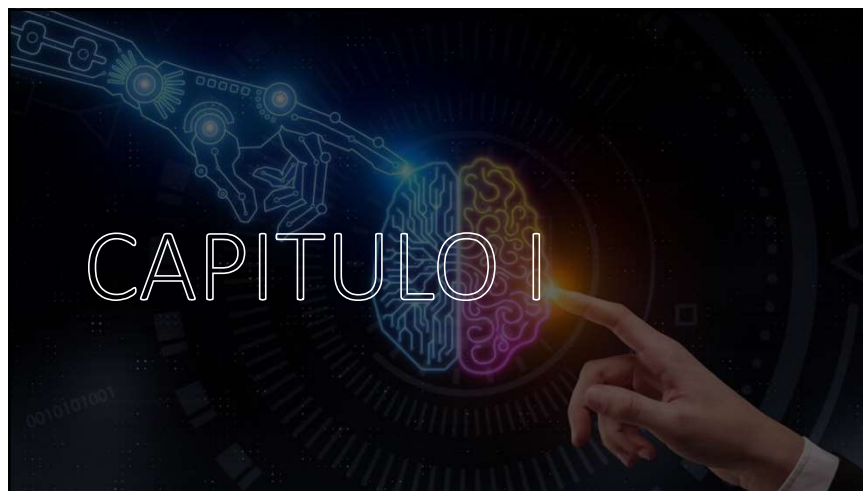
Anexos 2: Presentaciones



1



3



2

La siguiente tabla permite visualizar una comparación entre un sistema clásico y un sistema experto

Sistema clásico	Sistema experto
Conocimiento y procesamiento combinados en un programa.	Base de conocimiento separada del mecanismo de procesamiento.
No contiene errores	Puede contener errores
No da explicaciones, los datos sólo se usan o escriben	Una parte del sistema está formado por el módulo de explicación
Hacer cambios es tedioso	Los cambios en las reglas son fáciles de hacer
El sistema sólo opera completo	El sistema puede funcionar con pocas reglas
Se ejecuta paso a paso	La ejecución usa heurísticas y lógica
Necesita información completa para operar	Puede operar con información incompleta
Representa y usa datos	Representa y usa conocimiento

4



Herramientas y técnicas para el desarrollo de sistemas inteligentes

Las herramientas y técnicas para el desarrollo de sistemas inteligentes son variadas y la integración de ellas permite lograr sistemas de alto nivel. En general, el proceso de desarrollo de sistemas expertos se realiza en función de la naturaleza del sistema que se construye, la estrategia de desarrollo y las herramientas de apoyo.

5

Ventajas de los Shells y los productos de Sistemas Inteligentes

Los sistemas construidos con estos métodos útiles sin ningún conocimiento específico del dominio de aplicación se conocen como **shells** (conchas), sistemas esqueléticos o simplemente herramientas de IA.

Sin embargo, es conveniente aclarar que los shells no ayudan a la adquisición de conocimiento, pues esto se refiere a la tarea de proporcionarle conocimientos al sistema experto, la cual es realizada por los ingenieros del conocimiento

7

Fases del Desarrollo del Sistema

- Finalización del Proyecto
- Análisis y diseño de sistemas
- Prototipado rápido
- Desarrollo del sistema
- Implementación
- Post- implementación

6

Alternativas de desarrollo de Sistemas Inteligentes

El desarrollo de sistemas inteligentes que aporten soluciones a problemas de ingeniería y ciencias físicas, biológicas y computacionales, ha permitido que surja la necesidad de utilizar métodos más sofisticados.

El soft-computing consiste en una variedad de paradigmas de computación que busca incorporar el conocimiento humano de manera eficiente, manejar la imprecisión y la incertidumbre y el aprender a adaptarse a ambientes desconocidos o cambiantes para la obtención de un mejor desempeño. Existen cuatro técnicas principales: Redes Neuronales, Sistemas difusos, razonamiento probabilístico, algoritmos evolutivos.



8

Aplicaciones de los Sistemas Inteligentes y la IA

- Contenido (Percepción, Procesamiento de lenguaje natural, aprendizaje y desarrollo, planificación, variedad de razonamiento, estudio de las representaciones, técnicas y mecanismos de memoria, sistemas multiagente, mecanismos afectivos, robótica, búsqueda, ontologías).
- Técnicas (Medicina, robótica, diversos aspectos de la ingeniería, interfaces y sistemas de "ayuda", educación, gestión de la información, matemáticas, industrias del entretenimiento, biología, leyes, comercio, espacio, actividades militares).

9

Guía evaluativa de herramientas para crear Sistemas Inteligentes

surge la necesidad de utilizar un método de evaluación para determinar cuáles son las herramientas adecuadas y así seleccionar la herramienta que mejor se adapte a los requerimientos de los procesos que el sistema inteligente va a realizar. La siguiente guía pretende orientar en la evaluación de esas herramientas.

- Consiste en cinco elementos que se enumeran a continuación:
- Características de la aplicación
- Capacidades de la herramienta
- Métricas
- Técnicas de evaluación
- Contextos

11

Aunque los sistemas expertos se desarrollen con el propósito de dar solución a problemas, es preciso enumerar las capacidades y limitaciones de estos tipos de sistemas:

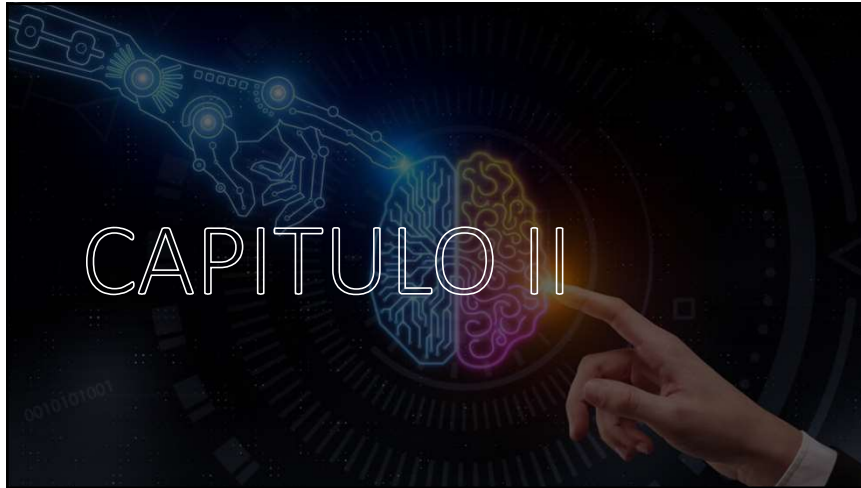
Capacidades	Limitaciones
✓ Asesoramiento	✓ Sustitución de personas encargadas de tomar decisiones
✓ Instrucción y ayuda a humanos en la toma de decisiones	✓ Poseer capacidades humanas
✓ Demostración (teoremas u otros)	✓ Generación de una salida precisa utilizando un conocimiento insuficiente
✓ Diagnóstico (médico u otros)	✓ Mejora de su propio conocimiento
✓ Razonamiento	
✓ Interpretación de entrada	
✓ Predicción de resultados	
✓ Justificación de una conclusión	
✓ Sugerencia de opciones a un problema	

10

Pasos para evaluar herramientas para crear Sistemas Inteligentes

- Paso 1: Determinar las características de la aplicación
- Paso 2: Identificar los contextos relevantes
- Paso 3: Derivar las capacidades significativas de la herramienta
- Paso 4: Identificar las métricas discriminantes y las técnicas de evaluación
- Paso 5: Identificar las herramientas disponibles
- Paso 6: Filtrar las herramientas disponibles para identificar a las herramientas candidatas
- Paso 7: Podar y priorizar cada una de las dimensiones
- Paso 8: Aplicar el esquema de la estructura para evaluar y seleccionar herramientas

12



13

¿Qué es Expert System Builder?

Expert System Builder o ES-Builder es una herramienta gratuita que permite la creación de sistemas simples para la enseñanza de inferencias y construcción de bases de conocimiento. Fue desarrollado por McGoo Software con fines principalmente educativos y es una versión web mejorada con el marco de desarrollo AJAX. Esta versión se basa en ES-Builder 3.0 que era una aplicación para Windows.



15

Capítulo II: Desarrollo de aplicaciones con Expert System Builder

La creación de sistemas expertos requiere de un software o herramienta que permita insertar fácilmente el conocimiento de uno o más expertos humanos de modo que se pueda dar solución a distintos problemas en un dominio en particular.

14

Entre las características del ES-Builder se mencionan:

Permite la visualización instantánea y vista previa del sistema experto como un sitio web.

Marca los errores en ramas inválidas del árbol de decisión para asistir al usuario que desarrolla el sistema experto y así brindarle asistencia en la identificación de errores en el árbol.

El usuario puede hacer copias de seguridad y restauración de su proyecto mediante una interfaz web.

Permite eliminar ramas enteras en lugar de eliminar las partes no deseadas del árbol de decisión.

Se puede copiar y pegar ramas enteras para reestructurar el árbol de decisión y ayudar en el desarrollo más eficiente de este.

16

Etapas de desarrollo del Expert System Builder

Antes de empezar el desarrollo de un sistema experto en ES-Builder se debe definir el tema y el dominio al que este pertenece, se deben establecer el alcance del tema, es decir, los límites. Posteriormente se procede a crear la especificación, la cual consta de dos pasos:

- Identificar
- Crear una especificación
 - Identificación
 - Especificación
 - Diseño
 - Documentación
 - Implementación del sistema
 - Pruebas
 - Evaluación
 - Presentación

17

Capítulo III: Desarrollo de aplicaciones con Exsys Corvid

Exsys Corvid, también llamado Corvid, es una poderosa herramienta de propósito general ampliamente utilizada para desarrollar aplicaciones interactivas de sistemas expertos.

Está dirigida a aquellas personas que no están familiarizadas con la programación por lo que es de fácil aprendizaje y provee la flexibilidad necesaria para el manejo de resolución de problemas tanto básicos como complejos.

19



18

Las principales características con las que cuenta esta herramienta son las siguientes (Exsys Corvid, 2011):

- Su ambiente de trabajo intuitivo permite a los expertos del dominio fácilmente describir de forma lógica los pasos para la toma de decisiones.
- Utiliza diagramas lógicos de tipo árbol que permiten describir secciones individuales del proceso.
- Permite la obtención de datos a través de métodos como el encadenamiento hacia adelante y el encadenamiento hacia atrás, lo que posibilita dividir el problema en partes más pequeñas para un rápido desarrollo estructurado.
- Provee dos tipos de vistas de la lógica: 1) diagramas de árbol que permiten al usuario ver la estructura completa del sistema y 2) un texto de las reglas individuales. El principal beneficio es que, en lugar de ver líneas de código, los usuarios pueden ver la lógica de una manera fácil de comprender que permite a los desarrolladores del sistema editar rápidamente las reglas.
- El motor de inferencia de Exsys Corvid combina y analiza las reglas para determinar las piezas que se necesitan y cuáles reglas pueden utilizarse para obtener las conclusiones deseadas.

20

es preciso mencionar los tres problemas principales en el desarrollo de sistemas expertos que se resuelven con el uso de Exsys Corvid

- Captura de la lógica de toma de decisión: Exsys Corvid provee múltiples formas de describir la lógica a través del uso de reglas IF-THEN basadas en variables.
- Construcción de una interfaz de usuario: Exsys Corvid permite que esta sea una tarea simple, por lo que no se requiere de conocimientos en programación en HTML para uso del sistema experto en línea.
- Integración con otros recursos de tecnología de la información: Para que un sistema experto funcione de manera eficiente necesita que se integre con otros recursos para obtener datos, guardar resultados o monitorear procesos

21

En este sentido, se pueden mencionar las siguientes áreas de aplicación de esta potente herramienta (Exsys Corvid, 2015):

23

Áreas de aplicación de la herramienta

Tanto corporaciones como compañías pequeñas utilizan esta herramienta. Los tipos de sistemas expertos más comunes que se construyen con Corvid son de (Exsys Corvid, 2011):

- Diagnóstico
- Mantenimiento (predictivo o solución de problemas)
- Cumplimiento normativo
- Recomendación de productos
- Atención al cliente
- Análisis de datos

22

Construcción de una aplicación con Exsys Corvid, 2011.

Determinar el problema específico que el sistema va a resolver: Describir de manera precisa el problema en particular al que el sistema brindará una solución para evitar confusión en pasos posteriores.

Recopilar la documentación de la lógica del sistema: Documentar la lógica de la toma de decisiones y los pasos requeridos para resolver el problema, de manera precisa, para que sea más fácil construir el sistema.

Determinar la arquitectura para el sistema: Seleccionar aspectos técnicos como el encadenamiento hacia adelante o hacia atrás, determinación de dónde y cómo manejar operaciones de procedimiento, segmentación de la lógica en bloques reusables, entre otros factores.

Utilizar Corvid para describir la lógica del sistema: Convertir la documentación del paso 2 en reglas utilizando Corvid.

Diseñar la interfaz del usuario final: Diseñar una interfaz de usuario que produzca los resultados correctos de forma atractiva. El diseño de la interfaz en las primeras fases permite obtener resultados óptimos, ya que se puede probar durante la construcción del sistema.

Probar el sistema: Hacer pruebas al igual que se hace cuando se desarrolla un proyecto de desarrollo de software. Los resultados deben ser validados por el experto del dominio y las reglas del sistema deben imprimirse para que sean revisadas individualmente por el o los desarrolladores del sistema en Corvid.

Implementar el sistema: Una vez validado el sistema, implementarlo en el servidor y ponerlo a disposición de los usuarios finales.

24

Primer paso: Elección de un problema y descomposición en pasos lógicos

Desarrollar un sistema experto en Corvid no sólo requiere determinar el problema al que se le dará solución, pues también es importante tomar en cuenta si al problema se le puede dar una solución a través del uso un sistema basado en el conocimiento.

Pantalla de variables

Las variables son los elementos clave en Corvid ya que se utilizan para describir el proceso de toma de decisión, hacer preguntas y mostrar resultados.

• Área de formato de variables

Esta sección de la pantalla de variables brinda la posibilidad de configurar las propiedades de las variables. Las pestañas que más comúnmente se utilizan son las siguientes: Prompt, Options, Ask with.

25

Definiendo valores de las variables Corvid

La mayoría de los sistemas desarrollados en Corvid están contruidos con variables de tipo estática y numéricas para definir las reglas de tipo IF, mientras que los dos últimos tipos de variables se utilizan para hacer recomendaciones.

• Variables de tipo lista estática

Son variables que tienen una lista específica de múltiples valores y puede haber cualquier cantidad de valores en la lista.

• Variables numéricas

Son variables a las que se les asigna un valor numérico y puede obtenerse preguntándole al usuario, a través del cálculo realizado por las reglas, por medio de un recurso externo, etc.

• Variables de confianza

Son variables a las que se les asigna un valor que indica el grado de certeza en un resultado específico o recomendación.

27

Primer paso: Elección de un problema y descomposición en pasos lógicos

• Área de lista de variables

Las variables que se utilizarán en el sistema aparecen como una lista en el cuadro de la parte izquierda de la pantalla de variables.

• Área de valores de las variables

Dependiendo del tipo de variable, el valor de esta se puede preguntar al usuario para que ingrese el dato o bien, se puede asignar el valor.

• Añadiendo variables

Para agregar una nueva variable en el sistema, se debe hacer clic en el botón "New"

26

Definición y representación gráfica

Como se mencionó anteriormente, la ventana "New Variable" ofrece la opción de agregar un tipo de variable denominada variable de confianza, la cual permite medir el grado de incertidumbre de la respuesta o solución que el sistema proporcione al problema en cuestión.

28

Ventana de comandos

La ventana de comandos “Display Commands”, se compone de:

- El área “Commands” que despliega una lista de los comandos en la parte superior.
- Bajo el área de comandos se encuentra una pequeña ventana que muestra el comando que se está editando o construyendo.
- En la parte inferior está una serie de controles que permiten construir y dar formato a los comandos mostrados en el área “Commands”.

29

Bloques de control

Los bloques de control son los que permiten añadir los bloques lógicos y los nodos para las reglas IF-THEN. Los controles que admiten estas opciones se mostraron anteriormente, por lo que en esta sección se explicará cómo agregarlos al sistema.

- Añadiendo los bloques lógicos

Antes de añadir los bloques lógicos es necesario crear las variables que se utilizarán en el sistema experto.

- Añadiendo las condiciones IF

Las condiciones IF se muestran como un grupo relacionado de nodos que proveen condiciones para una misma variable.

- Adición de nodos ENTONCES

Los nodos ENTONCES o THEN asignan un valor a una variable y se agregan debajo de cualquiera de las condiciones IF y estos se indican por medio de flechas de color amarillo

31

Segundo paso: Añadir bloques lógicos

Corvid ofrece una manera única para definir, organizar y estructurar las reglas del sistema en bloques lógicamente relacionados. Estos son grupos de reglas que pueden definirse a través de diagramas de árbol o como reglas individuales.

Cada bloque contiene varias reglas y árboles o bien, una sola regla. El bloque contiene reglas que se relacionan con un aspecto específico de la tarea de toma de decisión.

30

Tercer paso: Añadir bloques de comandos

Un bloque de comandos o “Command Block” en Corvid se encarga de decirle al sistema qué hacer, a diferencia de los bloques lógicos que indican el cómo hacer las cosas. Se pueden mencionar los siguientes usos más comunes para estos tipos de bloques:

- Determinación del tipo de encadenamiento para la ejecución de las reglas.
- Control del orden en el que se ejecutan los bloques.
- Obtención de datos de programas externos al iniciar la sesión.
- Envío de resultados y recomendaciones del sistema a otros programas.
- Ejecución de un conjunto de reglas varias veces para los ciclos FOR o WHILE.
- Despliegue de ventanas complementarias como ayuda o para propósitos de explicación.
- Visualización de reportes.
- Envío de correos.

32

Cuarta etapa: Ejecución

Para ejecutar un sistema desarrollado en Corvid se debe hacer clic en el ícono del botón "Run". Existen varias formas de ejecutar un sistema, pero por defecto Corvid utiliza el Exsys Corvid Applet Runtime que es una forma de ejecutar un programa de Java como parte de una página web.

33

Capítulo IV: Desarrollo de aplicaciones con Visual Prolog

En este capítulo se explicará el lenguaje de programación Prolog el cual está basado en el paradigma de la programación lógica. Un paradigma representa una filosofía para diseñar soluciones y en este caso sería aplicando lenguajes de programación.

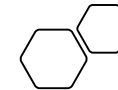
La programación lógica puede entenderse entonces como un paradigma de programación basado en la lógica de primer orden, también llamada lógica de predicados, y puede verse como una deducción controlada.

35



34

Características del lenguaje

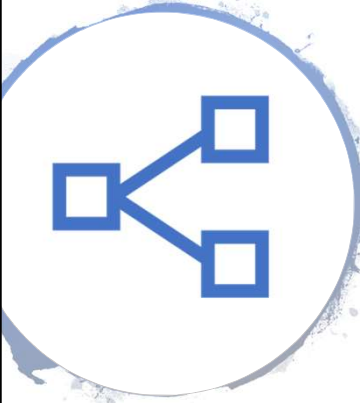


Prolog permite un prototipado más rápido que con muchos lenguajes porque es más próximo a la especificación lógica del programa. Se utiliza en la computación simbólica y no-numérica y permite resolver problemas que involucran objetos y relaciones entre los objetos, lo cual incluye:

Análisis de estructuras bioquímicas

- Bases de datos relacionales
- Comprensión del lenguaje natural
- Lógica matemática
- Resolución de problemas abstractos
- Resolución de ecuaciones simbólicas
- Diversas áreas de la inteligencia artificial

36



Fundamentos de Prolog

Según (Clocksin & Mellish, 2012) la programación en Prolog consiste en tres principios básicos que son:

- Especificar algunos hechos acerca de los objetos y sus relaciones.
- Definir algunas reglas acerca de los objetos y sus relaciones.
- Hacer preguntas acerca de los objetos y sus relaciones.

37

Del lenguaje natural a los programas Prolog

La sintaxis de Prolog es fácil de aprender, pero parte del aprendizaje de esta es comprender cómo se puede pasar del lenguaje natural al lenguaje Prolog. En esta sección se explicará un ejemplo que permita entender cómo llevar conocimiento del lenguaje natural al lenguaje Prolog.

- **Cláusulas: Hechos y reglas**
Los hechos y reglas se denominan cláusulas y son los hechos los que le dan sentido a un programa en Prolog
- **Predicados (relaciones)**
Como se indicó en la sección anterior de este capítulo, las relaciones se denominan **predicados** y este corresponde a una palabra o término que permita representar la relación entre objetos.
- **Variables (cláusulas generales)**
Anteriormente en este capítulo, se describió el uso de las variables en Prolog y se mencionó que estas pueden ser utilizadas para escribir las reglas y las consultas. Más adelante se explicará su uso en las consultas para una mejor comprensión de estas.
- **Objetivos (consultas)**
Una vez se tengan las cláusulas, es posible obtener conocimiento de la base de hechos a través de las consultas o preguntas.
- **Comentarios**
Parte de las buenas prácticas de programación es el uso de comentarios y en Prolog los comentarios pueden ayudar a que, tanto el programador como personas ajenas al desarrollo de este, comprendan el uso de las cláusulas y variables

39

PROgramando en LOGica

Como todo lenguaje de programación, para utilizar Prolog es imprescindible conocer la sintaxis, recordando que los elementos básicos de este lenguaje son los hechos, las reglas y las preguntas o consultas. Estos se explicarán a través de ejemplos para una mejor comprensión de la sintaxis de este lenguaje.

- Sentencias: Hechos y reglas
Un hecho es una proposición que puede ser cierta o falsa y es el que establece una relación entre objetos.
- Consultas
Una vez se han definido los hechos es posible hacer preguntas acerca de estos. La sintaxis para escribir una consulta o pregunta en Prolog es similar a la utilizada para escribir un hecho, con la diferencia de que se utiliza el símbolo “?-” antes de escribir el hecho que se quiere consultar.
- Variables: Sentencias generales
En Prolog no sólo se pueden nombrar objetos particulares, sino que también se pueden utilizar términos conocidos como variables para representar objetos que no se pueden nombrar

38

Programas Visual Prolog

Visual Prolog es un lenguaje de programación basado en Prolog y con un enfoque basado en la combinación de los paradigmas de programación lógica, funcional y orientada a objetos. Entre las principales características de Visual Prolog se pueden mencionar:

- Basado en la programación lógica con las cláusulas de Horn, que es un sistema formal para el razonamiento acerca de objetos y sus relaciones (Mims, 2008).
- Completamente orientado a objeto.
- Utiliza patrones de emparejamiento y unificación.
- Bases de datos de hechos totalmente integradas.
- Soporta el manejo automático de memoria (Smith, 2004).
- Soporta conexiones con otros lenguajes de programación como C, C++, HTML y Java, además de incluir funciones de Windows (Smith, 2004).

40

Entorno de desarrollo Prolog

El entorno de desarrollo de Visual Prolog o Integrated Development Environment (IDE) de Visual Prolog está fundamentalmente diseñado para facilitar el desarrollo, pruebas y modificación de aplicaciones. Entre las principales facilidades que se pueden mencionar están:

- Representación en forma de estructura de árbol donde se muestran los módulos. Además se incluyen archivos y recursos en la ventana "Project".
- El editor de texto que permite editar y buscar declaraciones e implementaciones.
- El editor de diálogo que provee controles para el diseño de cuadros de diálogo.
- El editor de menú que permite la creación de menús.
- El editor de la barra de herramientas que permite crear diferentes tipos de barras de herramientas.
- El editor de gráficos que permite la creación, vista y edición de íconos, cursores y bitmaps.

41

Unificación y backtracking

El intérprete de Prolog utiliza dos procedimientos para hacer inferencias y deducir nueva información a partir del conocimiento expresado en la base de conocimiento: la unificación y el backtracking.

El mecanismo de unificación: El mecanismo de unificación o matching es el proceso mediante el cual las variables toman valor en Prolog. A través de este es posible obtener respuestas a las consultas que se formulen y por ello la unificación constituye uno de los mecanismos esenciales de Prolog.

Backtracking: El proceso de backtracking o reevaluación consiste en volver a ver lo que se ha hecho e intentar resatisfacer los objetivos a través de una forma alternativa.

Predicados preddefinidos: Los predicados preddefinidos en Prolog son aquellos que ya están definidos y por lo tanto, no hay necesidad de especificarlos a través de cláusulas.

Prolog desde un punto de vista procedural: Prolog se diferencia de otros lenguajes de programación, como BASIC, Pascal y C, en que es declarativo.

43

Secciones básicas de un programa Visual Prolog

A continuación se detalla en qué consisten cada una de estas secciones.

- **La sección "clauses":** Esta sección se compone de cláusulas e inicia con la palabra clauses. Un archivo puede contener más de una de estas secciones y tiene como función especificar implementaciones de los predicados o valores iniciales de los hechos.
- **La sección "predicates":** En esta sección se declaran los objetos del ámbito actual y un archivo puede contener más de una sección de estas.
- **La sección "domains":** En Prolog existen cuatro tipos de variables: unsigned (números no negativos), integer (valores enteros), real (valores reales), character y string (cadenas).
- **La sección "goal":** La sección "goal", o sección de objetivos, se compone esencialmente de una lista de subobjetivos que son iguales al cuerpo de una regla.
- **La sección "database":** Esta sección corresponde a la base de datos de hechos, también denominada sección "facts". Los hechos describen los valores de los atributos de un objeto o clase.
- **La sección "constants":** Se sabe que una constante es un valor que no varía durante la ejecución de un programa.
- **Secciones globales:** Visual Prolog permite la declaración de dominios, predicados y cláusulas del programa de forma global.
- **Directivas de compilación:** Las directivas de compilación en Visual Prolog pueden añadirse al programa para indicarle al compilador formas específicas en las que debe tratar al código durante el proceso de compilación.

42



44

Capítulo V: Desarrollo de aplicaciones con CLIPS

En los capítulos anteriores, se entró en detalles sobre lenguajes de programación y entornos de desarrollo que permiten modelar el conocimiento humano de una forma más intuitiva.

Sin embargo, en este último capítulo se hará un mayor enfoque en la programación a través del lenguaje CLIPS, el cual es otro lenguaje que está diseñado para facilitar el desarrollo de aplicaciones para el modelado del conocimiento humano.

45



¿Qué es CLIPS?

Las primeras versiones solamente tenían capacidad para representar reglas y hechos. Sin embargo, a partir de la versión 6.0 empezó a ser posible incluir objetos en las cláusulas de las reglas.

También, se pueden utilizar los objetos sin las reglas a través del envío de mensajes, de manera que deja de ser necesario el uso de la máquina de inferencia si sólo se utilizan objetos

47

Introducción a CLIPS

CLIPS, que significa "C Language Integrated Production System", es un entorno de programación creado en 1984 por Software Technology Branch (STB), NASA, en el Johnson Space Center.

Este es un lenguaje se basa en el paradigma de programación basado en reglas, lo que o hace útil para la creación de sistemas expertos y otros programas en los que se requiera de una solución heurística para facilitar la implementación y mantenimiento de una solución algorítmica.

46

En la programación en CLIPS existen tres elementos básicos y uno adicional que no es básico, pero se considera importante:

Tipos de datos: Representan información y existen ocho tipos de datos primitivos:

- **integer:** Puede ser cualquier número con o sin signo seguido por dígitos. Es equivalente al tipo de dato long integer en C.
- **float:** Son números de tipo coma flotante o decimales. Es el equivalente a la coma flotante de doble precisión en C.
- **symbol:** Secuencia de caracteres que empieza con cualquier carácter ASCII y es seguido por un 0 o cualquier otro carácter ASCII. Símbolos como los paréntesis, el símbolo "&", la raya vertical "|", las doble comillas y el símbolo "" se denominan delimitadores y no deben incluirse en los símbolos. El símbolo "?" está reservado para el uso de variables.
- **string:** Es una serie de caracteres que comienzan con las doble comillas ("), seguido por un cero u otros caracteres y termina con doble comillas (").
- **external-address:** Es la dirección de un dato externo retornado por una función que se ha escrito en otro lenguaje y ha sido integrado con CLIPS. Sólo se puede crear este tipo de dato a través del llamado a una función y se representa con el formato <Puntero-XXXXXX>, donde XXXXX es la dirección externa.
- **fact-address:** Es la dirección o índice de un hecho y se representa con el formato <Hecho-XXX>, donde XXX es el índice del hecho.
- **instance-name:** Es el nombre de un objeto y se forma colocando un símbolo dentro de "[]".
- **instance-address:** Es la dirección de una instancia y se representa con el formato <Instancia-XXX>, donde XXX es el nombre de la instancia.

48

En la programación en CLIPS existen tres elementos básicos y uno adicional que no es básico, pero se considera importante:

Funciones: Manipulan los datos y deben escribirse entre paréntesis para hacer llamados.

- **Constructores:** Añaden conocimiento a la base de conocimiento y deben ir entre paréntesis. Estos nunca retornan un valor.
- **Comentarios:** Permiten documentar el código escrito en CLIPS y una línea de comentario inicia con el símbolo “;”.

49

En la programación en CLIPS existen tres elementos básicos y uno adicional que no es básico, pero se considera importante:

Funciones: Manipulan los datos y deben escribirse entre paréntesis para hacer llamados.

- **Constructores:** Añaden conocimiento a la base de conocimiento y deben ir entre paréntesis. Estos nunca retornan un valor.
- **Comentarios:** Permiten documentar el código escrito en CLIPS y una línea de comentario inicia con el símbolo “;”.

51

CLIPS presenta las siguientes características:

- Tiene un sistema de producción que incluye un sistema de mantenimiento de verdad con encadenamiento hacia adelante, adición dinámica de reglas y hechos y diferentes estrategias de resolución de conflictos.
- Es fácilmente integrable en aplicaciones de diferentes lenguajes y está disponible en diversas plataformas.
- Incluye COOL (Clips Object-Oriented Language) y extensiones para uso de lógica difusa o *fuzzy logic* (FuzzyCLIPS).
- Permite la integración con otros lenguajes de programación como C y Java. Además, CLIPS puede ser llamado desde un lenguaje de programación procedural y viceversa.

50

- **Hechos:** Representan información sobre el estado del mundo por lo que son un elemento de información fundamental. Pueden ser de dos tipos: ordenados y no ordenados.
- **Reglas:** Representan cosas que hacer con/sobre los hechos.
- **Agenda:** Conjunto de reglas.

Además, un programa básico escrito en CLIPS tiene tres componentes principales:

52

Elementos básicos de una herramienta de un Sistema Inteligente

Existen tres características esenciales que convierten a CLIPS en una herramienta apropiada para desarrollar sistemas inteligentes:

- Posee un shell basado en reglas lo que permite que la base de conocimiento, que inicialmente está vacía, se pueda llenar con la información necesaria sobre el estado inicial del problema.
- Utiliza el encadenamiento hacia adelante, es decir, que a partir de una serie de hechos deriva otros hechos utilizando las reglas que coinciden con los hechos conocidos.
- El motor de inferencia usa internamente el denominado algoritmo RETE para la búsqueda de patrones, de manera que puedan encontrar reglas que coincidan con los hechos.

53

Elementos Básicos de Clip

Anteriormente en este capítulo, se mencionaron los elementos básicos de un programa en CLIPS. En esta sección se hará una explicación más detallada de dichos elementos para tener una mayor comprensión de la sintaxis de CLIPS.

- Hechos en CLIPS

Los hechos son los que le permiten a CLIPS conocer el estado del sistema. Estos pueden tener un solo campo o más de uno. Existen tres formas de expresar los hechos en CLIPS, para lo cual se utiliza el comando (assert).

- Formas de ver los hechos

Para hacer que CLIPS muestre los hechos que se han introducido con su correspondiente número de identificador se puede utilizar el comando (facts).

- Tipos de hechos: Tipos de campos

Con los ejemplos que se han visto hasta el momento, se han estudiado los hechos denominados hechos sin etiqueta, que quiere decir que los campos que componen el hecho no llevan ningún tipo de etiqueta que lo identifique.

- Introducción remota de hechos

Por otro lado, es importante mencionar que existe la posibilidad de introducir un conjunto de hechos de forma remota, lo cual se logra a través del uso del comando (deffacts).

55

Entrada y salida de CLIPS

Para comenzar a programar con CLIPS en Windows se debe hacer clic en el ícono que se muestra en la Figura 53, el cual a continuación abrirá una ventana con el cuadro "Dialog Window" o "Ventana de Diálogo", que actúa como interfaz de usuario.

54

Reglas en CLIPS

Las verdaderas responsables de almacenar el conocimiento del sistema experto son las reglas, ya que estas ejecutan las acciones determinadas cuando se cumplen una serie de condiciones que son las que realmente almacenan el conocimiento.

- Definición de reglas: Para definir reglas en CLIPS se utiliza el constructor (defrule).
- Activación de una regla: Una regla se compone de dos miembros:
 - El miembro izquierdo o antecedente que está formado por una serie de condiciones. Cuando todas las condiciones o hechos se cumplen, entonces la regla se activa y realiza las acciones del miembro derecho de esta.
 - El miembro derecho o consecuente son las acciones que suelen ser afirmaciones, eliminaciones de hechos u otras acciones distintas. Dicho de otra manera, son las acciones que se ejecutarán cuando la regla se active.

56

- La agenda

La agenda es una estructura que muestra las reglas que se han activado y que después van a ser ejecutadas, colocadas en orden de ejecución. Existen dos formas de ver la agenda:

- En FuzzyCLIPS, se puede ver la ventana de agenda eligiendo la opción Agenda Window del menú Window.
- En la versión para terminales de texto, se utiliza el comando (watch agenda), que muestra en el terminal cuando se ha activado la regla en cuestión

57

La cola de activaciones

CLIPS hace uso de una estructura de cola para el almacenamiento y operación con las reglas que se han activado en un momento dado. El mecanismo de activación de las reglas que utiliza CLIPS consiste en lo siguiente: CLIPS realiza un ciclo con todas las reglas en el orden en que éstas han sido introducidas, comprobando qué reglas cumplen las condiciones necesarias para ser activadas, es decir, que se cumplan todos y cada uno de los patrones que hay en su antecedente.

Cuando esto sucede, la coloca en cola, de modo que la última regla que se ha activado sería la última en ejecutarse.

59

Variables en CLIPS

Al igual que en otros lenguajes de programación, en CLIPS las variables se utilizan para almacenar valores, por lo que el contenido de una variable es dinámico porque puede cambiar dependiendo de los valores que se le asignen.

- **Variables en el antecedente de una regla**

Normalmente las variables se usan para almacenar un valor en el antecedente de una regla, para luego utilizarlo en el consecuente de dicha regla.

- **Variables que almacenan direcciones**

Una variable puede almacenar la dirección de un hecho y es útil cuando se quiere conocer la posición que ocupa en la memoria un determinado hecho que se ha introducido y se quiere eliminar.

- **Comodines: Monocampo y multicampo**

Algunas veces sólo se especifica una parte de un hecho y el resto de este puede contener cualquier valor o simplemente estar vacío.

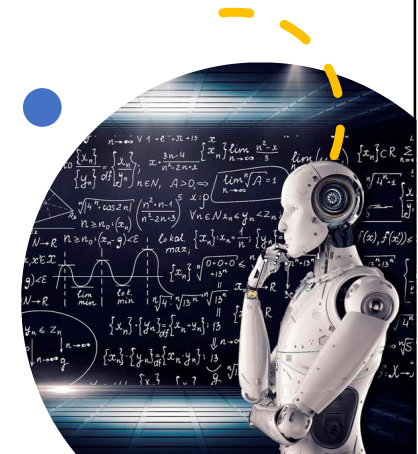


58

Hechos más complejos

Anteriormente, se mencionó que uno de los tipos de hechos, los denominados hechos con etiqueta o plantilla, son insensibles al orden. Estos consisten en una serie de campos que almacenan datos relativos a un determinado hecho. Dichos campos están etiquetados, por lo que no es necesario introducir valores en un orden predeterminado.

- **Definición de plantillas:** Las plantillas se pueden definir usando la instrucción (deftemplate).
- **Uso de plantillas en reglas:** Las plantillas se pueden utilizar en reglas de la misma manera que los hechos no estructurados.
- **Restricción ~:** Actúa negando el valor sobre el que actúa. Para comprender mejor su uso, se puede observar el siguiente ejemplo, en el que se quiere escribir una regla que niegue el paso a toda persona de la base de conocimiento que no tenga como nombre "Pedro"
- **Restricción |:** Se utiliza para combinar varios hechos. Por ejemplo, como se muestra a continuación, supongamos que se quiere definir una regla que permita el paso a "Pedro" y "Lola".
- **Restricción &:** Se utiliza para conectar varias restricciones en unión.
- **Operaciones aritméticas en CLIPS:** CLIPS proporciona las funciones y operaciones básicas para realizar operaciones aritméticas sobre datos de tipo numérico: +, -, *, /, div, max, min, abs, float e integer. Su significado es análogo al que tienen en otros lenguajes de programación.



60

Hechos más complejos

- **Operaciones aritméticas en CLIPS:** CLIPS proporciona las funciones y operaciones básicas para realizar operaciones aritméticas sobre datos de tipo numérico: +, -, *, /, div, max, min, abs, float e integer. Su significado es análogo al que tienen en otros lenguajes de programación.
- **Función de asignación Bind y Create\$:** La asignación de un valor a una variable en el antecedente de una regla mediante correspondencia de patrones es análogo a enlazar un valor a una variable en el consecuente usando la función (bind).
- **Read y Readline:** La función (read) se utiliza para leer información introducida desde el teclado o desde un fichero abierto previamente.
- **Otras características:** CLIPS tiene un conjunto de funciones lógicas y aritméticas predefinidas que pueden ser utilizadas para la comparación de números, variables y cadenas en el antecedente de una regla.