Universidad Central "Marta Abreu" de Las Villas
Facultad de Matemática- Física- Computación
Licenciatura en Ciencia de la Computación

TRABAJO DE DIPLOMA

UCSHELL 2.0: UN AMBIENTE PARA EL DESARROLLO DE SISTEMAS EXPERTOS

Autor: Lissett Fundora Fernández

Tutor: Dr. Mateo G. Lezcano Brito

El que suscribe: Lissett Fundora Fernández, hago constar que el trabajo titulado “UCShell 2.0, Un ambiente para el desarrollo de Sistemas Expertos ”, fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la universidad.

_____

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del tutor Firma del jefe del Laboratorio

_____

Fecha

No sigas el camino, ve por donde no haya vereda y deja una huella.

Anónimo

A mami, papi y Juli.

Agradecimientos

A Dios.

A mi familia toda, por ser impulso y guía.

A Julio, por mantenerse cerca aún en la distancia.

A mis amigos: los nuevos y los viejos, los que se fueron y los que están, por molestarme, alegrarme y soportarme. A Lisandra, Laudy, Lianet, Jorge L, Luis E, Ana Lilian, Lien, Emily, Yanet, Adis, Saimy, Mónica, Adis Perla, Brenda, David, Moreria, Joel, Marlon, Yeny, Michel y Mario.

A mi tutor Dr. Mateo Lezcano, por ser un evangelio vivo.

A los profesores maravillosos con los que he tenido la suerte de contar, especialmente a la profesora Dra. Zenaida García, por ser ese ángel, vigilante y presente, que nunca me dejó sola.

A Pavel, Inti, Moreira y Mario, por prestar sus maravillosos talentos para responder tantas dudas.

A Isel y Gonzalo por la ayuda con la interfaz visual.

A la vida, que me ha dado tanto

Resumen

UCShell, es un producto informático para el desarrollo de sistemas expertos. La versión 1.0 del sistema se desarrolló en el lenguaje Borland Pascal y se implementó específicamente para el sistema operativo MS-DOS. Para programar la segunda versión se usó el lenguaje Object Pascal y a su nombre original se le agregó, al inicio, la letra W para significar el hecho de que estaba específicamente diseñado para el sistema operativo Windows.

En este trabajo se presenta la versión 2.0 de UCShell (Shell de la Universidad Central) que retoma su nombre original y se significa el hecho de que puede ejecutarse sobre cualquier sistema operativo al estar programada en el lenguaje Java.

Las versiones anteriores del sistema han tenido un amplio uso en la docencia e investigación pero adolecen de algunas facilidades que se incorporan o mejoran en la versión actual, entre ellas: el cálculo de la certidumbre, la interfaz con el usuario y el módulo de explicación.

La nueva versión mejora de manera ostensible los mecanismos de inferencia atendiendo a que incluye una nueva dirección de búsqueda, la búsqueda dirigida por datos, e igualmente trae consigo una nueva acción llamada FINDALL que permite que continúe la inferencia una vez encontrado el resultado, lo que posibilita el hallazgo de valores alternativos de solución al problema.

Abstract

UCShell is a software product for developing expert systems. Version 1.0 of the system was developed in Borland Pascal language and implemented specifically for MS-DOS operating system. To program the second version was used the Object Pascal language and to its original name was added, at the beginning, the letter W to denote the fact that it was specifically designed for the Windows operating system.

In this paper is presented the version 2.0 of UCShell (Shell Central University) that sums up its original name and it means that you can run it on any operating system because it is programmed in the Java language.

Earlier versions of the system have been widely used in teaching and research but they lack some facilities that incorporate or enhance the current version, including: the calculation of certainty, the user interface and explanation module.

The new version improves, in an obvious way, the inference mechanisms bearing in mind that it includes a new searching address, data-driven search, and also brings a new action called Find All that allows continuing the inference once results are found, which makes possible the find of alternative values for solving the problem.

Introducción

Los Sistemas Expertos, considerados un tipo de Sistema Basado en Conocimiento, son sistemas que pretenden razonar de una forma similar al ser humano, restringiéndose a un espacio de conocimientos limitado. En teoría pueden razonar siguiendo los pasos que seguiría un experto humano (médico, analista, empresario, etc.) para resolver un problema concreto. Este tipo de modelo de conocimiento por computadora, ofrece un extenso campo de posibilidades en resolución de problemas y en aprendizaje.

El grupo de Inteligencia Artificial de la UCLV ha creado diferentes herramientas para construir sistemas basados en el conocimiento. Como una extensión de ese trabajo el laboratorio de Informática Educativa ha desarrollado varios sistemas computacionales para el construir Sistemas Expertos, entre ellas cabe citar la máquina de inferencia UCShell que fue implementada en el lenguaje Pascal sobre el Sistema Operativo MS-DOS; una segunda versión de esa máquina de inferencia denominada WUCShell fue implementada para el Sistema Operativo Windows y se programó en el lenguaje Object Pascal.

La gramática para UCShell V 1.0 se definió de acuerdo a las reglas del generador de compiladores YACC, por el contrario para WUCShell se implementó de forma manual, lo que trae por consecuencia que el compilador sea muy difícil de modificar debido, sobre todo, a que no existe una documentación adecuada del mismo.

Las versiones existentes del software solo permiten realizar inferencia en dirección backward, de ahí que no dan la posibilidad de inferir óptimamente sobre Sistemas Expertos que hayan sido diseñados para realizar inferencias dirigidas por datos.

Objetivos Generales

1. Obtener una nueva versión de WUCShell programada en Java, generando su compilador en forma automatizada, lo que permitirá hacerle modificaciones y ampliaciones de una manera más fácil.

2. Lograr una implementación que pueda ejecutarse sobre cualquier plataforma.

Objetivos Específicos

1. Mejorar los procesos de inferencia.
2. Documentar los módulos del sistema.
3. Implementar, en Java, un software profesional que permita editar, compilar, ejecutar y poner a punto el programa escrito por el usuario.

Preguntas de investigación

1. ¿Qué cambios habrá que hacerle a la nueva versión de UCShell de tal forma que se pueda modificar al software en el futuro de manera más fácil y productiva?
2. ¿Qué mejoras serán necesarias implementar al mecanismo de inferencia para que funcione de una forma óptima?

Justificación

En los último años el desarrollo de la inteligencia artificial ha sido vertiginoso, los sistemas basados en el conocimiento, aunque no tan novedosos, se han mantenido como una de las líneas de investigación de los grupos que trabajan en el campo de la IA. La UCLV ha sido puntera desde hace varios años en el desarrollo de herramientas computacionales que permiten construir sistemas basados en el conocimiento.

Tradicionalmente la construcción de los sistemas mencionados ha estado muy enfocada hacia plataformas de la familia de sistemas operativos de la Microsoft, WUCShell fue y es una herramienta exitosa lo que se demuestra por su uso en la docencia en la UCLV y otras universidades cubanas y extranjeras. La primera de las máquinas vinculadas a este proyecto tuvo el nombre UCShell y la segunda agregó la letra W al nombre de la anterior para significar el hecho de que era una versión sobre Windows. La nueva versión retoma el nombre original tratando de desligarse de cualquier sistema operativo, lo que permitirá que los sistemas obtenidos no estén atados a un sistema en particular.

Viabilidad

El grupo de Informática Educativa de la UCLV tiene una amplia experiencia en el desarrollo de herramientas para construir otros sistemas, en particular en el diseño, implementación y puesta a punto de sistemas de software que permiten el desarrollo de Sistemas Expertos y cuenta además con las plataformas adecuadas para llevar a la realidad el proyecto que se plantea en esta investigación.

Hipótesis

El producto de software UCShell 2.0 presenta mejoras en el proceso de inferencia con respecto a sus versiones anteriores y cuenta con una vasta documentación que recrea todas las etapas de su desarrollo e implementación.

Capítulo I. Sistemas Expertos

I.1 Objetivos del capítulo

Este capítulo tiene como propósito mostrar qué son los Sistemas Expertos, su importancia y uso en diferentes esferas de la sociedad. Se describen además algunos conceptos relacionados con los mismos, que son importantes para comprender su evolución e implementación. Se usará el estilo Harvard para referencias bibliográficas.

I.2 Inteligencia Artificial

La Inteligencia Artificial (IA) es una de las ramas de las ciencias de la computación que más se ha desarrollado en los últimos años. Las investigaciones en esta área, datan del principio de la década de los 50, pero es en el año 1956 cuando se comienza a usar este término, el mismo se le atribuye a John McCarthy. Generalmente, la IA es el área de las ciencias en la que se desarrollan técnicas para permitir a las computadoras actuar de la misma manera que lo haría un organismo inteligente. Los objetivos varían desde los extremos más débiles, donde un programa parece "un poco más inteligente" de lo que cabría esperar, hasta el extremo más fuerte, donde el intento es desarrollar una entidad basada en computadora completamente consciente e inteligente. El extremo inferior está continuamente desapareciendo en el marco general de la informática de acuerdo a la evolución que van teniendo el hardware y el software (Raynor 1999).

Algunas de las técnicas usadas para construir programas que tengan un comportamiento inteligente son las Redes Neuronales Artificiales (RNA), los Algoritmos Genéticos (AG), el Manejo de la Incertidumbre y los Sistemas Basados en el Conocimiento (SBC). Estos últimos son aquellos en los cuales se tiene un conocimiento específico del dominio que facilita el desarrollo de largas etapas de razonamiento, permitiendo así resolver casos recurrentes en dominios de conocimiento restringidos. No obstante, para resolver un problema en la práctica necesitan saber de antemano la correspondiente respuesta (Norvig 2004).

I.3 Los Sistemas Expertos

Los Sistemas Expertos (SE) son programas que resuelven problemas de un dominio de aplicación concreto de manera similar a como lo haría un experto humano en esa materia. Deben ser capaces de explicar las conclusiones y el razonamiento subyacente

(Bratko 1990). Por otra parte un experto humano es un individuo que posee un entendimiento superior del problema. A través de la experiencia el experto, desarrolla las habilidades que le permiten resolver los problemas de manera efectiva y eficiente (Gutiérrez 1998). Dentro de los Sistemas Basados en Conocimiento, los Sistemas Expertos se destacan por los éxitos alcanzados. Estos sistemas están compuestos por dos elementos básicos:

- La **Base de Conocimiento** (BC), que contiene el conocimiento sobre el problema.
- La **Máquina de Inferencia** (MI), que implementa los métodos para manipular dicho conocimiento.

Los conceptos anteriores se ilustran en la Figura I.1.

La segunda se encarga de inferir nuevos conocimientos, a través de un determinado mecanismo y utiliza con ese fin el contenido de la primera que expresa, en un formalismo dado, el conocimiento acerca de un dominio específico.

Podemos ver entonces al Sistema Experto como un modelo:

$$SE = BC + MI \qquad I.1$$

La BC almacena el conocimiento en una determinada notación, conocida como Forma de Representación del Conocimiento (FRC) que constituye el formalismo de este modelo.

La MI, por su parte, implementa los Métodos de Solución del Problema (MSP). Desde este punto de vista un SE es entonces:

$$SE = FRC + MSP \qquad I.2$$

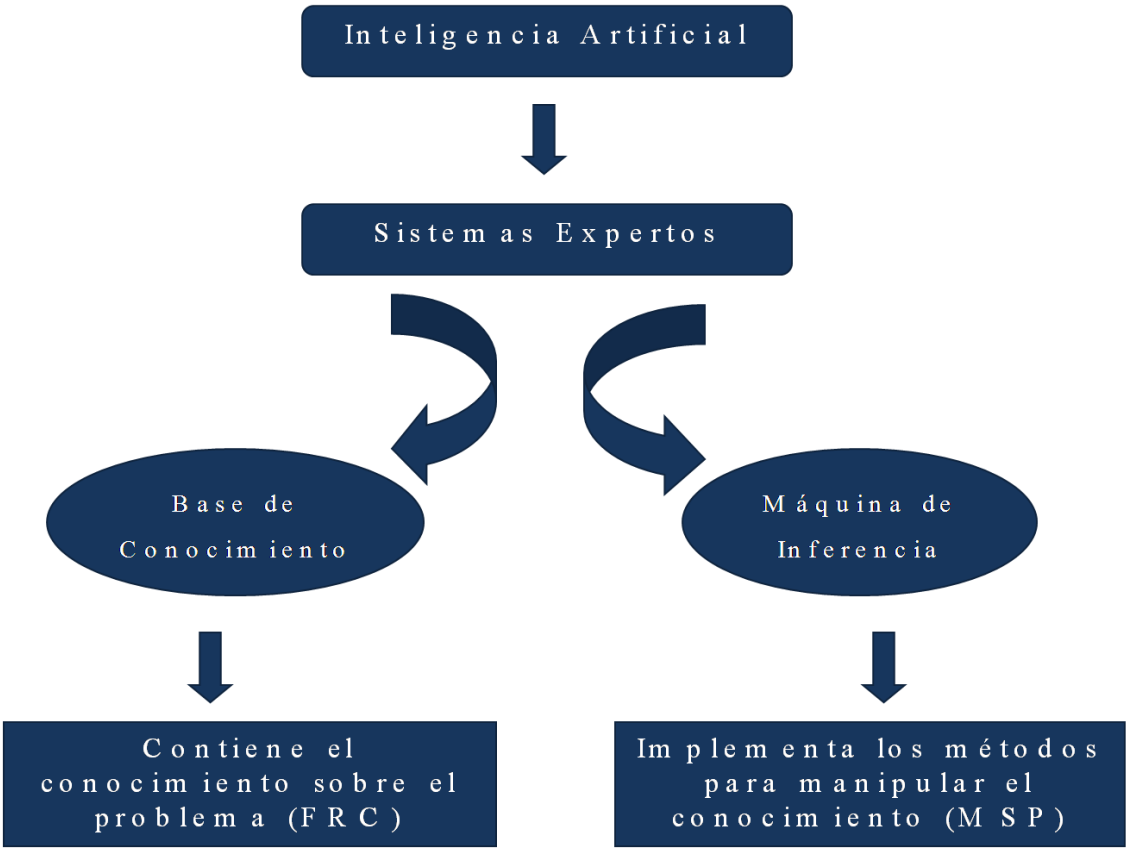

Figura I-1 Representación esquemática de Sistemas Expertos

I.4 Tipos de problemas que enfrentan

Los problemas que enfrentan los Sistemas Expertos pueden clasificarse, en consecuencia a la naturaleza de las situaciones para las cuales están diseñados como (Gutiérrez 1998):

- **Deterministas:** Los problemas de este tipo pueden ser modelados usando un conjunto de reglas que relacionen varios objetos bien definidos. Los Sistemas Expertos que tratan problemas deterministas son conocidos como Sistemas Basados en Reglas porque sacan sus conclusiones basándose en un conjunto de reglas utilizando un mecanismo de razonamiento lógico.
- **Estocásticos:** En situaciones inciertas es necesario introducir algunos medios para tratar la incertidumbre. Por ejemplo, algunos Sistemas Expertos usan la misma estructura de los sistemas basados en reglas, pero introducen una medida

para la incertidumbre de la regla y de sus premisas. En este caso se pueden utilizar algunas fórmulas para calcular la propagación de la incertidumbre.

I.5 Formas de representación del conocimiento

El tema de la Representación del Conocimiento está asociado a las diferentes formas en que se organiza y procesa la información necesaria para apoyar el proceso de razonamiento inteligente (Partridge 1996).

Los diferentes tipos de Sistemas Basados en Conocimiento o Sistemas Expertos se definen por la forma en que se representa el conocimiento que lo compone, así como el método de inferencia mediante al cual se realiza el razonamiento. Algunas de estas formas de representar el conocimiento son: los Sistemas Basados en Frames (SBF), los Sistemas Basados en Casos (SBC), los Sistemas Basados en Probabilidades (SBP), Redes Neuronales (RN), Sistemas Basados en Modelos (SBM) y los Sistemas Basados en Reglas (SBR).

No existe actualmente una Forma de Representación del Conocimiento (FRC) general que permita usarse en todo tipo de aplicación y ante un problema de un dominio específico es necesario realizar la selección de la FRC más adecuada para dicho problema.

I.5.1 Sistemas Basados en Reglas

Las Reglas de Producción son la FRC utilizada por el sistema SESE. Estas fueron introducidas en 1943 por Post y es una de las más antiguas técnicas para representar un dominio de conocimiento en un Sistema Experto, siendo además una de las más naturales y sigue siendo ampliamente usada en Sistemas Expertos tanto prácticos como experimentales (George F. Luger and Stubblefield 1998).

Un SBR provee una estructura que resulta de fácil comprensión para la lógica humana. Los SBR son Sistemas Basados en Conocimiento, en los que la forma de representación del conocimiento empleada son las reglas de producción o reglas de inferencia lógica o simplemente reglas IF THEN, y como método de inferencia utiliza la regla modus ponens (Bello 2002).

Una regla de producción consta de un par ordenado (A, B), representado en el Cálculo Proposicional como A $\Rightarrow$ B, donde A representa el antecedente y B el consecuente de la regla.

Una regla de producción se interpreta como "si se satisface el antecedente, entonces se cumple el consecuente". Esta manera de interpretar la regla permite considerarla como una unidad relativamente independiente de conocimiento.

Las reglas de producción pueden adoptar varias formas:

- Si condición P entonces conclusión C.

  Ejemplo. I.1

  IF

  (P >= 180)

  THEN

  C := 'Pesado'

- Si condición P entonces acción A.

  Ejemplo I.2

  IF

  (P = 100)

  THEN

  ACTION Find A

- Si condición P entonces no C.

  Ejemplo I.3

  IF

  (P < 180)

  THEN

  C := 'no es pesado'

Los antecedentes de las reglas, independientemente de la forma que estas adopten, pueden ser simples o compuestos. Los compuestos se forman uniendo varias condiciones simples por medio de conectivas lógicas.

De acuerdo con (Lezcano 2000) algunas de las ventajas de los Sistemas Basados en Reglas son:

- Modularidad: Los SBR son altamente modulares. Cada regla es una unidad de conocimiento que puede ser añadida, modificada o eliminada, independientemente de las otras reglas existentes. Esto da flexibilidad al desarrollo de la base de conocimientos.
- Uniformidad: Todo el conocimiento del sistema se expresa en el mismo formato.
- Naturalidad: Las reglas son un formato natural para expresar conocimiento en algunos dominios.

I.6 Manipulación del conocimiento representado

Para la manipulación del conocimiento representado en reglas de producción se usan los llamados Sistemas de Producción (SP).

Un SP consta de tres componentes básicos:

1. **Base de datos (BD):** Se utiliza como memoria de trabajo y sirve para almacenar los datos iniciales e intermedios.
2. **Conjunto de reglas:** Las reglas de producción operan sobre la memoria de trabajo. Cada regla tiene una condición, que se satisface o no por los datos de la BD. Típicamente una regla tiene una parte izquierda (las condiciones) que tiene que ser verdadera de acuerdo a la información existente en la memoria de trabajo para poder usar su parte derecha (las conclusiones).El que la parte izquierda de una regla se satisfaga puede establecerse por referencia a la BD o preguntando al usuario del sistema de producción.
3. **Intérprete:** Tiene como función llevar a cabo el proceso de inferencia. La dirección de la búsqueda de una solución puede seguir diferentes estrategias:

- ***Dirigido por datos*** o encadenamiento hacia delante (forward chaining): Consiste en buscar reglas que se cumplan a partir de hechos conocidos que están almacenados en la BD, o sea, a partir de condiciones probadas agrega nuevos hechos o realiza acciones que están expresadas en las conclusiones de las reglas que se cumplen.

- ***Dirigido por objetivos*** o encadenamiento hacia atrás (backward chaining): Dado un objetivo a probar, busca las reglas que contienen ese objetivo en su conclusión para posteriormente tratar de establecer la verdad de sus condiciones.

- ***Una combinación de los dos anteriores:***

Los esquemas anteriores se enfrentan al conflicto que surge cuando hay más de una regla que puede ser seleccionada, por supuesto que no se trata de aplicarlas todas a la vez, de ahí que surja la necesidad de tener una Estrategia de Resolución de Conflictos que seleccione la regla que se debe aplicar. Se pueden citar diversas estrategias:

1) Establecer orden en los datos.

2) Clasificar las reglas por prioridad de ejecución.

3) Ejecutar la regla más recientemente instanciada o la última.

Según (Rossel 2010) el algoritmo de control que ejecuta el intérprete cuenta, en general, con cuatro pasos que se aplican iterativamente:

- ***Extracción:*** obtener el conjunto de reglas plausibles de ser utilizadas, esto dependerá de la estrategia empleada. En *forward chaining* se elegirán aquellas que tengan su condición satisfecha en la base de datos global, mientras que en *backward chaining* se elegirán aquellas cuya conclusión esté en el objetivo o subobjetivos actuales.

- ***Refinamiento:*** Se refinan las reglas obtenidas (y las pendientes de pasos anteriores si las hubiera) eliminado algunas.

- ***Selección:*** Se elige una regla del conjunto refinado (es posible elegir más de una, incluso alguna estrategia plantea elegir todas y no refinar). La selección aplica lo que se conoce como *resolución de conflictos.* Se

considera que hay un conflicto en el sentido de que varias reglas pueden ser aplicadas en ese momento.

- ***Aplicación:*** La aplicación de la reglas seleccionadas para producir modificaciones en la base de datos global.

Se dice que los sistemas que utilizan *forward chaining* son sistemas que usan una representación en *espacio de estados.* Pueden verse como un problema de moverse por distintos estados, donde cada estado corresponde con determinada configuración de la base de datos global. Por otro lado, los sistemas que razonan en forma *backward chaining* utilizan representación por *reducción de problemas,* partiendo de un problema a ser resuelto y reduciendo la complejidad del problema inicial a subproblemas, en forma iterativa.

En un esquema clásico se recorren todas las reglas buscando una que se satisfaga, posteriormente se aplica y esa acción que, en muchos casos, provoca cambios en la BD. El conjunto de reglas se sigue recorriendo hasta que se soluciona el problema (se alcanza el objetivo) o no es posible invocar más reglas, lo que hace que el sistema termine.

Una dificultad con la FRC denominada reglas de producción, es que, a medida que crece el número de reglas, crece el conocimiento del sistema y se hace más difícil la búsqueda. Una solución parcial a este problema es descomponer la BC inicial en varias partes que puedan ser procesadas de manera independiente. Además de descomponer la BC es necesario descomponer, por supuesto, la condición de terminación en varias sub-condiciones, una para cada sub-base. Un caso especial de esta descomposición es expresarla como una conjunción de las sub-condiciones componentes.

Los sistemas de producción que permiten descomponer su BC y la condición de terminación reciben el nombre de **Sistemas de Producción Descomponibles**, SESE implementa una sentencia (**CHAIN**) que lo permite.

I.7 Tratamiento de la incertidumbre

En el desarrollo de un proceso de razonamiento intervienen, al menos, tres elementos:

1. El conocimiento sobre el dominio de aplicación.

2. Un método para procesar el conocimiento.

3. Ciertas observaciones vinculadas con el objeto de razonamiento.

El razonamiento con incertidumbre denota un proceso de razonamiento en el que alguno de estos elementos no es totalmente preciso, o sea, el razonamiento se realiza sobre la base de una información parcial y, por lo tanto, los resultados de este son sugeridos pero no asegurados por las premisas del mismo.

A diferencia de la IA, el software algorítmico tradicional no puede lidiar con información incompleta, ya que si algún dato es incorrecto, la respuesta será incorrecta.

I.7.1 Fuentes de Incertidumbre

La presencia de incertidumbre en los sistemas de razonamiento se puede originar por varias fuentes, entre ellas se pueden mencionar:

- Imprecisiones en la definición de los conceptos y sus relaciones.
- Imprecisiones y pobre seguridad de los instrumentos usados para hacer las observaciones.
- Imprecisiones del lenguaje de representación en el que se trasmite la información.
- Falta de idoneidad de un formalismo para representar cierta clase de conocimiento.
- Agregación de información desde múltiples fuentes.
- Falta de seguridad en cuanto a, si un elemento dado pertenece a un conjunto bien definido, o en su pertenencia parcial a un conjunto cuyas cotas no están definidas rigurosamente.
- El dominio relevante es realmente aleatorio.
- El dominio relevante no es aleatorio dada la suficiente cantidad de datos, pero el programa no siempre tendrá acceso a todos esos datos.

Durante el razonamiento, la incertidumbre proveniente de estas fuentes se combina produciendo resultados parciales y finales que tienen su propia incertidumbre. A este proceso se le llama propagación de la incertidumbre.

Las representaciones numéricas son propias para representar y manipular la incertidumbre dada por la información imprecisa, pues ellas permiten trabajar con valores de confidencia. Es posible, además, definir un cálculo que ofrece un mecanismo para propagar la incertidumbre a través del proceso de razonamiento, sin embargo, la representación numérica no puede ofrecer una explicación clara de las razones que conducen a una conclusión dada.

Los modelos basados en este enfoque están, en su mayor parte, diseñados para manipular el aspecto de la incertidumbre derivado de la información incompleta pues, al carecer de medidas para cuantificar los niveles de confidencia, son inadecuados para manipular la información imprecisa y son más apropiados para permitir seguir la traza desde las fuentes de información a las conclusiones.

I.8 Desarrollo de un Sistema Experto

Antes de empezar a desarrollar un Sistema Experto es preciso determinar el área del saber para la cual se va a crear. En esta etapa se hace imprescindible la presencia de una persona o grupo de personas con vastos conocimientos en la materia los cuales quedarán vertidos finalmente en la base del sistema.

I.8.1 Ingeniería del conocimiento

La **Ingeniería del Conocimiento** es la disciplina relacionada con la forma en que se organizan, construyen y verifican estos datos para formar la Base de Conocimiento del Sistema Experto. De acuerdo a (Edward Feigenbaum 1983) la Ingeniería de Conocimiento es la disciplina que comprende la integración del conocimiento a los sistemas de computación con el objetivo de resolver problemas que normalmente necesitarían de un elevado nivel de experiencia humana.

La información que brindan los expertos en el dominio de la materia usualmente está llena de tecnicismos y resulta de difícil comprensión, es en este momento del desarrollo donde interviene el **Ingeniero del Conocimiento (IC)**. Este no es más que el que toma el conocimiento de un especialista y de una forma sencilla y útil lo transmite a una base de conocimiento. Los ingenieros del conocimiento son conocidos por su

capacidad para simplificar la información y las instrucciones de los expertos, su utilidad está dada en que ellos pueden organizar e interpretar la información interna para hacer sistemas de decisiones (Ruth Aylett 2002). Hay ciertas características que han de ser inherentes al Ingeniero del conocimiento:

- Debe ser capaz de conocer y comprender los Sistemas Expertos.
- Tener un buen dominio de las herramientas para crear los SE.
- Ser un buen programador.
- Un hábil entrevistador.

El proceso de extracción, codificación y verificación del conocimiento de un experto humano, llevado a cabo por el IC, se conoce como **Adquisición del Conocimiento**. Si se usan Reglas de Producción como forma de representación del conocimiento, la extracción se refiere a la formulación de las reglas, la codificación, a la escritura de las mismas en una determinada sintaxis y la verificación al refinamiento de la BC. La adquisición del conocimiento es el "Cuello de Botella" de las aplicaciones de los SE.

I.8.2 Traductores, compiladores e intérpretes

Una vez editada la base de conocimientos y antes de comenzar el proceso de inferencia es preciso saber que está bien conformada y además pasarle un código a la máquina de inferencia para que pueda hacer el proceso deductivo. El **Traductor,** se encarga de explorar la base de conocimiento, informar los errores sintácticos y semánticos y, si no hay errores, generar el código.

Un Traductor es un programa que toma como entrada un programa escrito en un lenguaje de programación (lenguaje fuente) y produce como salida un programa en otro lenguaje (lenguaje objeto). El traductor se escribe en un lenguaje denominado lenguaje de implementación.

I.8.2.1 Compilador

Cuando el lenguaje fuente es de alto nivel (Pascal, C++, etc.) y el objeto es de bajo nivel (lenguaje ensamblador), al traductor se le denomina **Compilador**. La estructura general de un compilador se muestra en la Figura I.2. A continuación se detallan algunos de los elementos que conforman la misma.

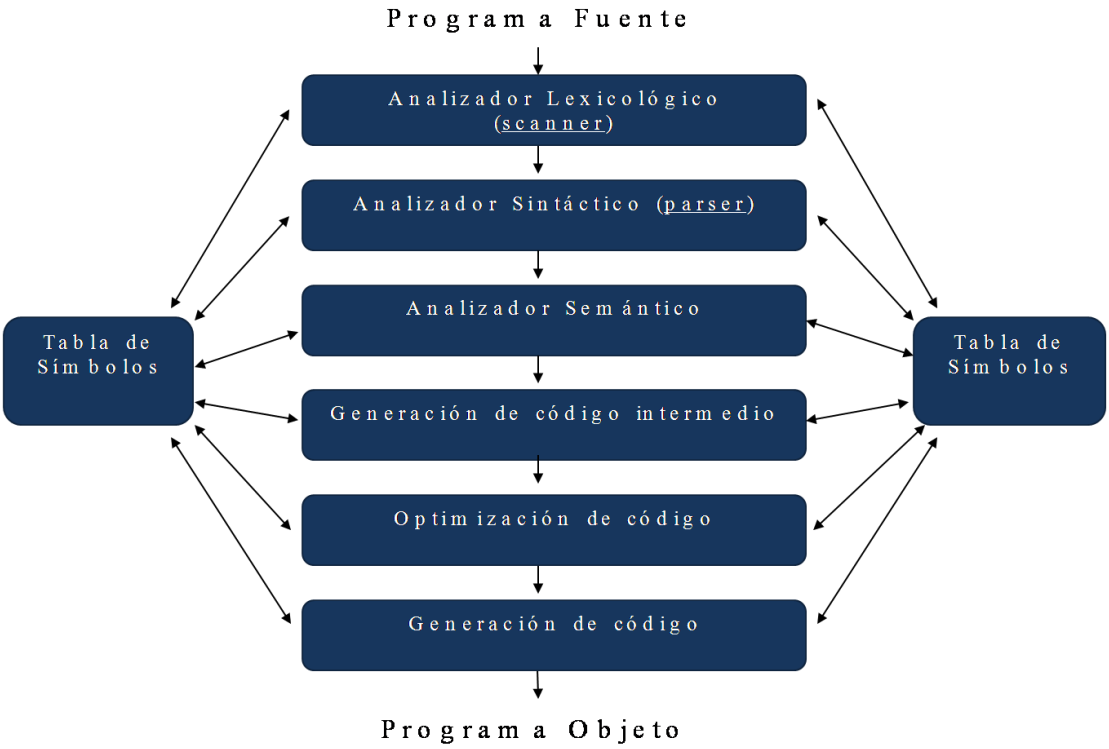

Figura I-2 Estructura general de un compilador

I.8.2.1.1 - Análisis lexicológico (scanner)

La entrada de un compilador es una secuencia de símbolos de un alfabeto. El analizador lexicológico o **Scanner** se encarga de tomarlos y agruparlos en entidades sintácticas simples o elementales denominadas ***tokens*** o ***lexemas***. Las categorías de los tokens pueden variar de un lenguaje a otro, pero en general se distinguen las siguientes:

- palabras reservadas
- identificadores
- constantes numéricas y literales
- operadores

A cada token se le asigna una estructura lexicológica consistente en un par de la forma <*tipo del token*, *info*>. La primera componente es una categoría sintáctica como "*constante*", "*identificador*", "*operador*", etc., y la segunda componente proporciona información relacionada con el token en particular (valor de la constante, índice del símbolo en la tabla de símbolos, etc.).

Se puede afirmar, por lo tanto, que el *scanner* es un traductor cuya entrada es una cadena de símbolos (programa fuente) y cuya salida es una secuencia de estructuras lexicológicas o *tokens*.

Ejemplo:

En la expresión: costo = (precio + imp) * 0.98

costo, precio e imp son *tokens* del tipo identificador;

0.98 de tipo constante;

= + * ( ) son *tokens* por sí solos.

Si se asume que todas las constantes e identificadores serán de la categoría de *tokens* id, la salida del *scanner* sería:

<id,1> <=, > < (, > <id,2> < +, > <id,3> <), > < *, > <id,4>

Las segundas componentes de los *tokens*, pueden ser, por ejemplo, los índices en una tabla de símbolos. Nótese que en algunos *tokens* la primera componente brinda toda la información necesaria sobre el mismo, por lo que no es necesario especificar una segunda componente. Luego, la secuencia anterior se puede representar de manera simplificada como:

$<id>_1 = (<id>_2 + <id>_3) * <id>_4$

I.8.2.1.2 Operaciones sobre la tabla de símbolos

Una tarea fundamental en un compilador es la de almacenar los identificadores utilizados en un programa y sus atributos principales, de manera que en cualquier momento pueda conocerse de un identificador, su tipo, alcance, etc., para el caso de los procedimientos, la cantidad y tipo de los parámetros, etc. Esta información se almacena generalmente en una estructura conocida como tabla de símbolos, la cual tiene una entrada para cada identificador y sus atributos. Los tokens que representan constantes o identificadores se almacenan en la tabla a medida que van apareciendo. En la Tabla 1.1 se muestra un ejemplo de una tabla de símbolos donde aparece el código de los tokens, el tipo de token y en caso de ser variable el tipo de la variable.

Tabla I-1 Ejemplo de tabla de símbolos

1	costo	variable	real
2	precio	variable	real
3	imp	variable	real
4	0.98	constante	real

Para otros tipos de identificadores se almacenan otros datos; para los identificadores de arreglos por ejemplo, se almacena el tipo de los elementos, las dimensiones, etc.

Esta tarea es muy importante, ya que durante todo el proceso de compilación se invierte una parte significativa del tiempo en el manejo de la tabla de símbolos. En la fase de análisis lexicológico se insertan los símbolos según aparecen en el programa fuente y en las restantes fases se van agregando atributos a medida que se conocen. Además, para cada identificador que se analice, es importante conocer sus atributos, por lo que el acceso a la tabla de símbolos se realiza constantemente. Aquí se hace evidente que las tablas deben organizarse de forma tal que permitan una búsqueda eficiente.

I.8.2.1.3 Análisis sintáctico (parsing)

La entrada del analizador sintáctico o parser es la secuencia de tokens generada por el scanner. El parser analiza solamente la primera componente de cada token; la segunda componente se utiliza en otros pasos.

El análisis sintáctico es un proceso en el cual se examina la secuencia de tokens para determinar si cumple ciertas convenciones estructurales de la definición sintáctica del lenguaje. En la Figura I.3 se muestra un ejemplo.

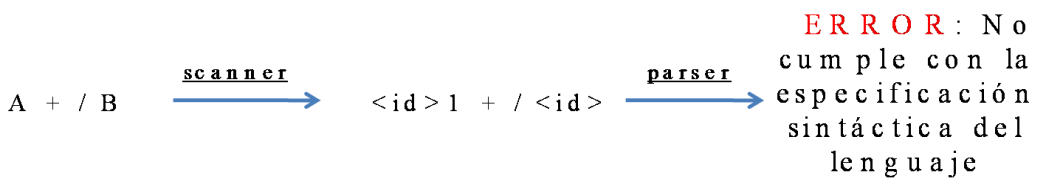

Figura I-3 Ejemplo de análisis sintáctico.

Es necesario conocer además, en el proceso de generación de código, cuál es la estructura sintáctica de una cadena. Por ejemplo, la estructura sintáctica de la expresión A+B*C debe reflejar el hecho de que B y C deben multiplicarse antes de que se realice la suma. Para ello se agrupan los tokens en una estructura en forma de árbol, conocida como árbol sintáctico.

Ejemplo. Árbol sintáctico para la secuencia:

<id>1 = (<id>2 + <id>3) * <id>4

Esta secuencia conduce a que se realicen las siguientes operaciones:

I.<id>2 se sume con <id>3

II.el resultado de I se multiplique por <id>4

III.el resultado de II se almacene en <id>1

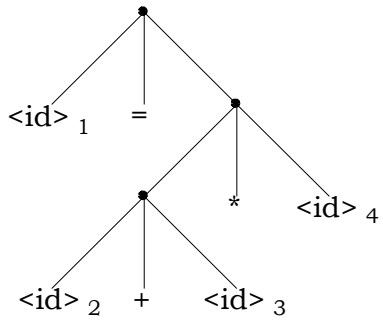

Figura I-4 Árbol sintáctico para la secuencia <id>1 = (<id>2 + <id>3) * <id>4

Los descendientes directos de cada nodo representan las acciones a realizar, y los valores para los cuales deben realizarse las acciones. Es necesario incluir los paréntesis en el árbol, ya que son utilizados para indicar el orden de evaluación en el programa.

I.8.2.1.4 Análisis semántico y Generación de código

El árbol creado por el parser se utiliza para generar la traducción del programa fuente y de esta forma lograr la interpretación de las acciones asociadas a la sentencia o sentencias en curso. La traducción puede ser a lenguaje de máquina, o a un lenguaje intermedio, como el lenguaje ensamblador. A partir del árbol sintáctico y de las tablas de símbolos se genera el código; sin embargo, la construcción del árbol y la generación de código se realizan en la práctica frecuentemente de forma simultánea.

El análisis semántico se hace para comprobar que las sentencias del programa tengan sentido. Uno de los chequeos semánticos más comunes es el chequeo de tipo. Ejemplo: en caso de que se tenga definida la operación *división* entre dos variables. En esta situación será necesario hacer un chequeo para comprobar que las variables que intervienen sean de tipo numérico ya que no tiene sentido realizar la división con cadenas.

I.8.2.1.5 Representación de lenguajes

En general existen dos esquemas diferentes para definir un lenguaje, los cuales se conocen como esquema generador y esquema reconocedor, en función del principio que se siga para la definición de las cadenas que pertenecen al Lenguaje. En el primer caso, o sea, en el caso de los esquemas generadores se trata de un mecanismo que permite "generar" las diferentes sentencias del lenguaje, en el segundo caso se trata de un mecanismo que permite reconocer si una cierta sentencia pertenece o no a un cierto lenguaje. Los representantes más significativos de estos esquemas son el **Esquema de Chomsky** como esquema generador de Lenguajes a través del concepto de **Gramática** (Gries 1971) y la **Teoría de Autómatas** (Alfred V. Aho 2006) como esquema reconocedor típico de Lenguajes.

Si un Lenguaje posee un número finito de cadenas, puede ser definido simplemente listando sus cadenas, pero ¿cómo proceder para definir lenguajes con un número infinito de cadenas? Para lograr esto se utiliza la Gramática.

En la definición de una Gramática se utilizan dos conjuntos disjuntos de símbolos denominados:

N: Conjunto de símbolos no terminales (utilizados para representar combinaciones de símbolos).

$\sum$: Conjunto de símbolos terminales

El centro de una gramática lo constituye un conjunto de "reglas de producción" o reglas de formación de cadenas, las cuales están formadas por elementos de la relación:

$$(N \cup \sum)^* N (N \cup \sum)^* \rightarrow (N \cup \sum)^* \qquad I.3$$

Definición: Una gramática es un cuádruplo G = {N, ∑, P, S}II donde:

N: Conjunto de símbolos no terminales.

∑: Conjunto de símbolos terminales.

P: Conjunto de Reglas de Producción.

S: Axioma o símbolo distinguido (S ∈ N)

Ejemplo de Gramática:

Sea la Gramática G1 = ({A, S}, {0, 1}, P, S) donde:

S → 0A1
0A → 00A1
A → **ε**

Como se puede apreciar una gramática define a un lenguaje en forma recursiva. Para denotar gramáticas en forma simplificada se acostumbra a utilizar la siguiente notación:

a, b, c, d, …	representan terminales
A, B, C, D, …	representan no terminales
α, β, λ, δ, …	representan cadenas de terminales y no terminales
u, v, w, x, …	representan cadenas de terminales

El concepto de lenguaje generado por una gramática puede ser precisado en la forma siguiente:

$L(G) = \{w \mid w \in \Sigma^* \wedge S \Rightarrow^* w\}$

Ejemplo:

$G^0 = (\{E, T, F\}, \{a, +, *, (,)\}, P, E)$

Donde:

P: E → E + T | T
T → T * F | F
F → (E) | a

Clasificación de Chomsky

Sea G = (N, Σ, P, S) una gramática, entonces:

1. Si cada producción en P es de la forma A → xB ó A → x con A, B ∈N y x ∈ Σ^*, entonces la gramática G se denomina **Lineal a la Derecha**.

2. Si cada producción en P es de la forma $A \rightarrow \delta$, donde $A \in N$ y $\delta \in (N \cup \Sigma)^*$, entonces la gramática G se denomina **Libre de Contexto**.

3. Si cada producción en P es de la forma $\alpha A \beta \rightarrow \alpha \delta \beta$, donde $A \in N$ y α, β, $\delta \in (N \cup \Sigma)^*$, entonces la Gramática G se denomina **Dependiente del Contexto**.

4. Si una gramática G no cumple las restricciones anteriores se denomina **Gramática sin Restricciones**.

I.8.2.2 Intérprete

En ocasiones en lugar de usar un compilador es necesaria la implementación de un Intérprete. Este es como un compilador, solo que la salida es una ejecución. El programa de entrada se reconoce y ejecuta a la vez. No se produce un resultado físico (código máquina) sino lógico (una ejecución) (Sergio Gálvez and Mata 2005).

Algunos de los motivos que provocan el uso de un intérprete en lugar de un compilador son los siguientes:

- Hay variables que toman como valor cadenas de caracteres que representan instrucciones del lenguaje fuente, y existen operadores que han de ejecutar el contenido de esas variables, información que no tendrán hasta el tiempo de ejecución.

- No se declaran las variables, de tal modo que estas tienen siempre el tipo del último valor que se le asignó.

- La presencia del intérprete durante la ejecución es necesaria por razones de seguridad o de independencia de la máquina.

- Cuando el lenguaje dispone de operadores muy potentes, lo que significa que la mayor parte del tiempo los programas están ejecutando código rápido prefabricado, más que los programas fuente del programador.

I.8.2.2.1 Ventajas y Desventajas

Entre las ventajas de un intérprete, se pueden mencionar:

- Flexibilidad: permite realizar acciones complejas, imposibles o muy difíciles con un compilador, como las siguientes:
    - Ejecución de cadenas de caracteres mediante operadores como "execute" o "interprete".
    - Modificar sobre la marcha el significado de los símbolos e incluso prescindir por completo de las declaraciones.
    - Obtener un ligamiento dinámico completo en los sistemas orientados a objetos.
    - Simplificar la gestión de memoria en los programas fuente.
- Facilidad de depuración de programas: la interpretación puede interrumpirse en cualquier momento para examinar o modificar los valores de las variables o la situación en la ejecución. La tabla de símbolos está disponible. Se pueden corregir los errores y continuar. Trazas y paradas programadas. Saltos en el programa. Abandonos de subrutinas.
- Rapidez en el desarrollo.

En cuanto a las desventajas de un intérprete, las más relevantes son:

- Velocidad: usualmente un orden de magnitud menor que la de un programa compilado.
- Tamaño del programa objeto, que exige añadir el intérprete al programa propiamente dicho.

I.9 Conclusiones del capítulo

En el capítulo se ha presentado un análisis de los Sistemas Expertos como un área importante de la Inteligencia Artificial. Disciplinas tales como la Ingeniería del Conocimiento han sido ilustradas y ha sido esclarecido el proceso de adquisición del conocimiento. Se han mostrado las diversas fases de creación de un sistema, desde la elección de la Forma de Representación del Conocimiento, la representación del lenguaje mediante la Gramática y las etapas por las que transcurre el programa desde que es escrito hasta que se compila o interpreta. En el próximo capítulo se mostrará una

visión interna del sistema así como la aplicación práctica de estos conceptos, y las vías escogidas para la puesta a punto del software.

Capítulo II UCShell 2.0. Visión Interna

II.1 Introducción

UCShell 2.0 es un ambiente que permite la implementación de Sistemas Expertos, a este le preceden las versiones anteriores UCShell 1.0 y WUCShell. UCShell 1.0 (Universidad Central Shell versión 1), se desarrolló para el Sistema Operativo MS-DOS usando el lenguaje Borland Pascal (Lezcano 1998). La segunda adoptó el nombre de WUCShell porque fue diseñada para Windows, se programó sobre el lenguaje Objectc Pascal. Para la versión actual se retomó el nombre de UCShell ya que esta no va a estar restringida a un Sistema Operativo en específico, teniendo en cuenta que ha sido programada en Java.

El sistema incluye las funcionalidades de las versiones anteriores y adiciona nuevos mecanismos de inferencia, trabajo con distintos valores de certidumbre así como una mejorada interacción con el usuario con respecto al manejo de errores.

En este capítulo se describen las herramientas computacionales utilizadas para la implementación del software y se presentan los diagramas que permiten entenderlo.

II.2 Módulos que componen el sistema

UCShell 2.0 se puede ver como un producto en general, pero en la práctica se divide en varios módulos. Aunque sus funcionalidades están muy relacionadas, cada uno de estos componentes puede funcionar independientemente de los demás. Estos son:

- ***UCShell Compiler 2.0:*** Este software permite compilar una base de conocimiento previamente creada, reconocer los errores en caso de que existan y generar la forma interna para realizar la inferencia en caso de que haya sido escrita correctamente. Para interactuar con el usuario cuenta con una sencilla interfaz visual.
- ***UCShell Symbolic Compiler 2.0***: Permite compilar bases de conocimiento con información simbólica. El análisis de errores que realiza es mucho menos profundo que el de UCShell Compiler 2.0 y además no reconoce los comentarios. La funcionalidad de este producto está dada en que permite

generar una forma interna a partir de la cual realizan inferencias productos de software tales como TeachShell[1].

- ***UCShell Library 2.0:*** Este software es una biblioteca que incluye los mecanismos de compilación de UCShell 2.0 y de UCShell Symbolic Compiler 2.0, además tiene incluida una máquina de inferencia. No cuenta con interfaz visual ya que fue diseñada para que otros productos hiciesen uso de los mecanismos que tiene incorporada y definieran su propia interfaz visual.
- ***UCShell IDE 2.0***: Un IDE (ambiente de desarrollo integrado) que permite la creación de proyectos y bases de conocimiento. Contiene un editor de programa con todas las funcionalidades habituales. Este software hace uso de la biblioteca UCShell Library 2.0 para ejecutar el proceso de compilación, tanto real como simbólica, e incluye los mecanismos de inferencia, la cual efectúa con componentes gráficos.

II.3 Actores y Casos de Uso

Para una mejor comprensión de la utilidad de cada uno de estos módulos se hace necesaria la descripción de los distintos tipos de usuario que los pueden utilizar y de los distintos casos de uso de cada uno específicamente.

Existen tres actores en el sistema: el primero es el "ingeniero del conocimiento, es el encargado de tomar el conocimiento del experto en la materia y escribirlo en forma de reglas de producción, usando la sintaxis definida para la base. El segundo usuario es el "programador", es el encargado de añadirle nuevas funcionalidades al software o modificar la interfaz visual. Finalmente se encuentra el "usuario común", es el que hará uso del sistema.

[1] TeachShell es un software para la enseñanza de Sistemas ExpertosMorales, Y. G. (2011). Implementación del software para la creación de Sistemas Expertos. Informática Educativa. Santa Clara, Universidad Central "Marta Abreu" de LAs Villas.

Para modelar el sistema se utilizó la herramienta Unified Modeling Language (UML). Este es un software ampliamente usado para la modelación y desarrollo del software.

A continuación se ilustran los diferentes casos de uso de cada uno de los módulos que componen al sistema.

Casos de uso del UCShell Compiler 2.0:

- Compilar base de conocimiento: El usuario compila la base de conocimiento. Si existe algún error de tipo sintáctico o semántico, que se pueda reconocer en tiempo de compilación, se mostrará en la línea en que ocurre. En caso contrario se generará la forma interna. Cualquiera de los actores descritos anteriormente puede realizar esta acción.

Casos de uso de UCShell Symbolic Compiler 2.0:

- Compilar base de conocimiento con información simbólica: Es similar a la anterior pero la forma interna se genera con información simbólica que usarán sistemas que trabajen con este tipo de información, tales como el TeachShell. Todos los actores descritos anteriormente puede realizar esta acción.

Casos de uso de UCShell Library 2.0:

- Compilar base de conocimiento: El usuario usa las clases de la biblioteca para realizar un proceso de compilación.
- Realizar inferencia: El usuario, toma una base de conocimiento, previamente compilada, y utiliza las clases definidas en la biblioteca para poner en marcha el motor de inferencia.
- Personalizar la interfaz visual: En caso de que se desee contar con una interfaz de usuario propia para el Sistema Experto, el programador, que es el actor capacitado para ejecutar esta acción, puede definir la forma en que se muestran los datos al usuario. El sistema cuenta con una ayuda que permite realizar estas operaciones de manera sencilla guiando al usuario con respecto a los cambios necesarios para lograr el objetivo.

Casos de uso de UCShell IDE 2.0:

- Editar base de conocimiento: Se podrán crear nuevos proyectos que agrupen varias bases de conocimiento o crear una de estas de manera independiente. En este último caso el IDE provee una plantilla genérica que facilita el trabajo del ingeniero del conocimiento. Se pueden abrir proyectos o archivos creados anteriormente. Se podrán editar los archivos con las facilidades propias de un editor de texto tales como las operaciones de copiado, pegado, guardado, búsqueda entre otras. Además el sistema, resalta las palabras reservadas del lenguaje para facilitar la programación.
- Compilación de la base de conocimiento
- Compilación de la base de conocimiento con información simbólica
- Realizar inferencia: La inferencia se realiza de manera interactiva ya que el sistema le puede hacer preguntas al usuario que se contestan seleccionando entre las posibles respuestas, en caso de que estas sean discretas o introduciendo el valor que se estime preciso de acuerdo a la interrogante del sistema. Una vez terminada la inferencia el usuario puede consultar, de manera visual, los resultados y ver de qué forma se alcanzaron.
- Consultar ayuda: El sistema cuenta con una ayuda que puede ayudar tanto al ingeniero del conocimiento a escribir la base, como al usuario común a interactuar con él.

En la Figura II.1 se muestran los actores y los casos de uso del IDE UCShell 2.0.

En la Figura II.2 se ilustra cómo se expande el caso de uso Realizar inferencia

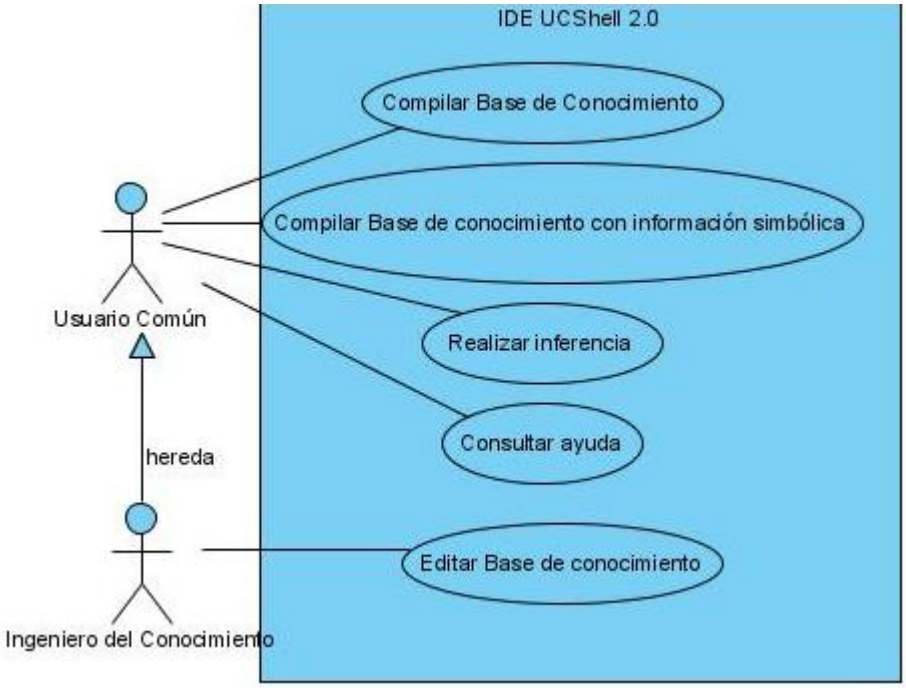

Figura II.1 Casos de uso y actores del IDE UCShell 2.0

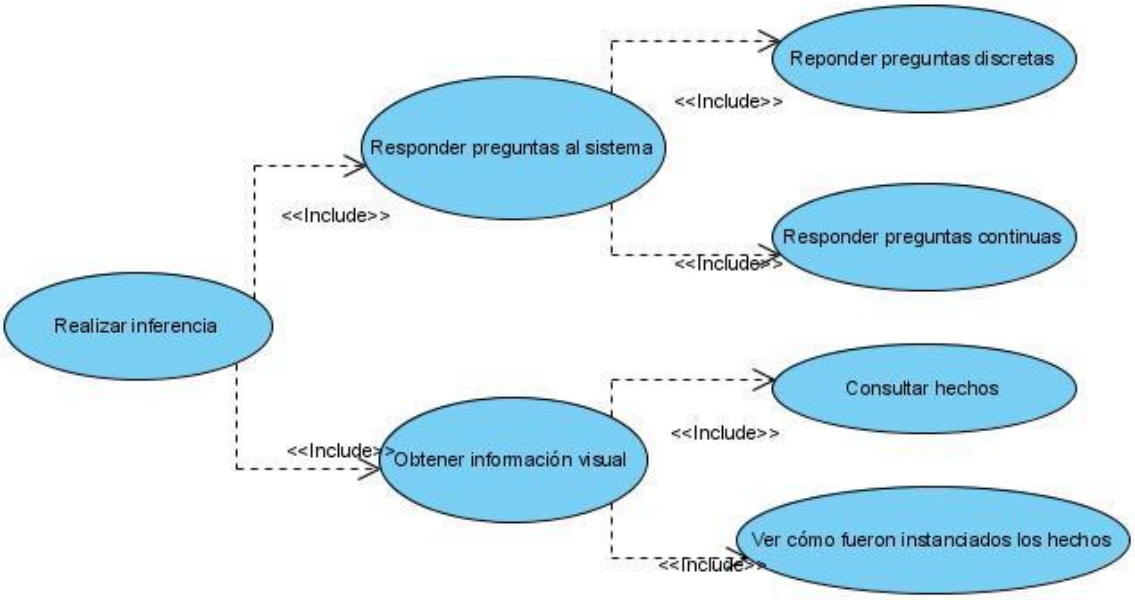

Figura II.1 Expansión del caso de uso realizar inferencia

II.4 Creación del sistema

El proceso de creación del sistema cuenta de varias etapas. Primero fue necesario definir la estructura que tendrían las sentencias de la base de conocimiento. Luego fue preciso reconocerlas y mostrar los posibles errores que podrían aparecer en la escritura. Una vez hecho esto hubo que implementar los mecanismos de inferencia mediante los cuales se efectúa la búsqueda. El lenguaje utilizado para cumplir estas metas fue Java, este es un lenguaje de programación de código abierto y las aplicaciones desarrolladas en el mismo pueden ejecutarse en cualquier plataforma. El ambiente de programación utilizado fue NetBeans en su versión 6.9 el cual es un IDE muy popular entre la comunidad de programadores de Java. Cada una de estas etapas de la creación de UCShell 2.0, así como las herramientas que fueron necesarias para su implementación, será descrita a continuación.

II.4.1 La Base de Conocimiento

Las Reglas de Producción fue la forma de representación del conocimiento escogida para el sistema, pero la base de conocimiento no se limita a las reglas. En la misma se pueden definir variables y acciones que guíen la inferencia según desee el Ingeniero del Conocimiento. La sintaxis establecida con este fin puede verse definida en cuatro bloques como muestra la Figura II.3.

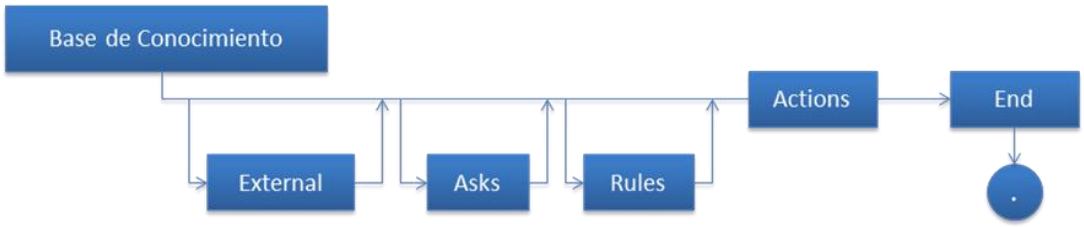

Figura II.3 Sintaxis de la Base de Conocimiento

Estos cuatro bloques son:

- **Variables externas**: en este bloque se declaran aquellas variables que pertenecen a otra base de conocimiento y se desea conservar su valor. Cuando a la base actual se arriba por una orden **CHAIN**, el sistema la carga y elimina la anterior, pero conserva las variables (con sus valores) que se hayan declarado en este bloque. El inicio de este se define por la palabra reservada **EXTERNAL** y contiene las variables que se quieren declarar externas, separados por coma.

- **Asks:** define las variables "preguntables". Este bloque está precedido por la palabra reservada **ASKS**. La declaración de un ask comienza por la palabra reservada **ASK** y continúa con los siguientes valores:
    - El **nombre de la variable**, este viene después de la palabra reservada ASK y a continuación del mismo se pone ":".
    - La **pregunta** que se le va a hacer al usuario. Esta ha de ser una cadena.
    - Una **lista de valores posibles** entre los cuales el usuario escogerá una respuesta. Esta lista estará encabezada por la palabra reservada **CHOICES** o **DOMAIN**. Los valores pueden ser también texto, imágenes o video, en este último caso la lista tiene que estar encabezada por DOMAIN, si las opciones son solo texto han de estarlo por CHOICES. Los valores se separan por comas. Si la cantidad de respuestas posibles no es discreta, como sería por ejemplo el peso de una sustancia, la interfaz presentada al usuario será un cuadro para que se escriba el valor. En cualquier caso las respuestas pueden estar acompañadas de un valor de certidumbre definida como un valor entre 0 y 1.
    - Un texto que muestra la razón por la que se hace la pregunta. Esta es una característica muy importante de los Sistemas Expertos, que consiste en explicarle **por qué** se hace la pregunta. Esta explicación ha de estar encabezada por la palabra reservada **BECAUSE**. El texto está compuesto por diferentes cadenas separadas por coma y es posible hacer referencia al valor que ha tomado una variable y la certidumbre. El valor de la variable se referencia "<variable>.ASK" y la certidumbre como "<variable>.CNF".

En la Figura II.4 se muestra un ejemplo de un Ask declarado dentro de una base de conocimientos.

```
ASKS
  ASK Reaccion_a :
   'Caliente el material seco en un tubo de combustión conteniendo en la boca'+
   'un papel de PH.'+
   '¿Cómo se comporta?'
  DOMAIN 'Débilmente alcalino','Neutro o débilmente ácido'
 BECAUSE
   'Porque estoy tratando de investigar si se trata de una Poliamida',
   'un Poliuretano o una Resina de Urea para lo cual esta reacción',
  'es determinante.'
```

Figura II.2 Ejemplo de Ask declarado dentro de una base de conocimiento

- **Rules**: Este es el bloque donde se definen las reglas de la base de conocimiento. Estas siguen la forma **IF**… **THEN**…, o sea, **SI** se cumple la condición **ENTONCES** se cumple la conclusión. Este bloque está precedido por la palabra reservada **RULES** y termina con la declaración del bloque de acciones (que no es opcional). La declaración de una regla comienza por la palabra reservada **RULE** y termina con "**END;**". Una regla internamente está compuesta por:
    - El **número** de la regla. Este es el identificador de la regla. En caso de que se declaren varias reglas con el mismo número el sistema mostrará que hay un error.
    - La **condición** de la regla. Comienza con la palabra reservada **IF**. El resultado de evaluar la misma ha de ser booleano, pero dentro de la regla se pueden declarar todos los tipos de operadores con que cuenta el sistema. La nueva versión de UCShell incorpora dos operadores nuevos, la potenciación (^) y el factorial (**!**).
    - Una lista de **conclusiones**. Comienza por la palabra reservada **THEN**. Es aquí donde, usualmente, toman valores las variables. Hay que notar que puede haber más de una conclusión en una misma regla y que cada una de estas se puede alcanzar con un valor de certidumbre distinto de 1. La forma de evaluación de la certidumbre se explicará más adelante. Una conclusión se declara como: el nombre de la variable seguido de "**:=**" y luego la expresión que se le asignará.

- Una lista de **acciones** que comienza por la palabra reservada **ACTIONS**. Estas se ejecutan en caso de probarse la regla. Usualmente en las acciones de una regla es donde se muestran los resultados que se alcanzaron durante la inferencia mediante la acción **DISPLAY**.
- La **explicación** de la regla. Permite explicar cómo se llegó a una conclusión. Su sintaxis es la misma que la de la explicación en los ask.
- La **certidumbre** de la regla. En el sistema se pueden definir reglas débiles.

En la Figura II.5 se muestra un ejemplo de una regla declarada en una base de conocimientos.

```
RULE 19
  IF (Familia      = 'Poliuretano')    AND
     (Rf           = 0.72)             AND
     (Color        = 'Amarillo pardo')
    THEN
      Composicion := '1-4_Fenilendiamina'
    ACTIONS
      DISPLAY 'Está formado por 1-4 Fenilendiamina.'
  END;
```

Figura II.3 Ejemplo de declaración de una regla dentro de una base de conocimiento

- **Actions:** Es el único bloque no opcional y define las acciones que ejecutará la máquina de inferencia. Todo el proceso de inferencia se inicia y finaliza en este bloque. Dentro de las acciones que pueden provocar el inicio de la inferencia están: **FIND**, **FINDALL** y **FINDFORWARD**. La descripción de cada una de estas acciones se efectuará más adelante. Este bloque está precedido por la palabra reservada **ACTIONS**.

El fin de la base de conocimiento se reconoce con la expresión "**END.**". En cualquier lugar de la misma se pueden poner comentarios al estilo de java. Estos serán ignorados por el compilador en el proceso de generación de código.

II.4.2 El compilador

Teniendo en cuenta que el número de sentencias de la base de conocimiento no es finito y que además el ingeniero del conocimiento es susceptible a cometer errores en la

sintaxis se hizo necesaria la confección de un compilador. En el caso de UCShell fue preciso construir un intérprete ya que el lenguaje es no tipado y no existe la declaración de variables, implicando esto que la creación de las mismas hay que hacerla en tiempo de ejecución y que el tipo que toman es el del último valor que se le asigna.

II.4.2.1 Análisis lexicológico (scanner)

Dentro del compilador, el scanner es el encargado de reconocer los distintos tipos de tokens de la gramática y analizar si estos están bien conformados, en caso de no ser así reportará el error y parará el análisis. Teniendo en cuenta que el sistema se quería implementar en Java, se utilizó JFlex que es un generador de analizadores lexicográficos desarrollado por Gerwin Klein como extensión de la herramienta JLex desarrollada en la universidad de Princeton. Este software además de generar código en Java también fue desarrollado en este mismo lenguaje (Sergio Gálvez and Mata 2005).

El scanner de UCShell 2.0, reconoce un conjunto de caracteres clásicos: operadores como el de la suma, la resta la multiplicación, la división etcétera. Con respecto a versiones anteriores se añadieron algunos nuevos para dar más posibilidades al trabajo matemático tales como la potenciación y el factorial. Además se define un conjunto de lexemas que serán palabras reservadas del lenguaje como los nombres de las acciones, los operadores trigonométricos y booleanos entre otros. La sintaxis del lenguaje reconoce los comentarios tanto de una línea como de varias, esa facilidad no estaba presente en las versiones anteriores. Las cadenas de caracteres se tomarán como literales si están entre comillas simples, los números se tomarán como enteros o como doubles si tienen parte decimal.

En el scanner se reconocen también variables. Estas pueden contener caracteres alfanuméricos, pero necesariamente han de comenzar en una letra. El sistema es sensible a mayúsculas, de ahí que por ejemplo: “perro” y “Perro” serán reconocidas como dos variables distintas.

II.4.2.1.1 Tratamiento de los errores

Los errores que reconoce el scanner son los de lexemas mal formados, por ejemplo una cadena, que no sea literal, y comience con un número seguido por letras. Cada vez que el analizador lexicográfico encuentra un lexema bien formado retorna su símbolo correspondiente. En caso de encontrar un error retornará el símbolo de error y como

consecuencia de esto se parará el proceso de compilación, se mostrará que hubo un error sintáctico y se le indicará al usuario la línea donde ocurrió el mismo.

II.4.2.2 La Tabla de Símbolos

La tabla de símbolos contiene las entradas para los distintos tipos de identificadores. Al inicio del análisis lexicográfico se agregan, a la tabla, todos los operadores permitidos en la gramática. Las variables se van insertando según van apareciendo. Tanto a las variables como a los operadores se les asigna un código para identificarlos internamente. La tabla de símbolos contiene una estructura con todos los datos necesarios para trabajar con cada entrada.

La tabla de símbolos fue implementada usando una tabla hash que tiene como entrada el nombre de la variable o la palabra reservada que lo identifica en el caso de los operadores.

II.4.2.3 El análisis sintáctico (parser)

Una vez realizado el análisis lexicológico es necesario examinar la secuencia de tokens para determinar si cumple ciertas convenciones estructurales de la definición sintáctica del lenguaje. Para hacer esto, el parser toma la secuencia de lexemas generada por el *scanner*.

En la construcción de UCShell 2.0 fue utilizado el generador de analizadores sintácticos CUP. Esta es una herramienta que fue desarrollada en el Instituto de tecnología de Georgia (Estados Unidos), la misma genera código en Java y permite introducir acciones semánticas escritas en este lenguaje.

II.4.2.3.1 La gramática

Teniendo en cuenta los requerimientos del lenguaje que se quiere representar, se hizo necesaria la definición de una gramática libre de contexto. En la misma se definieron un conjunto de 76 terminales y 37 no terminales (estos conjuntos son disjuntos). Dentro de los primeros se definen las palabras reservadas del lenguaje, los operadores y las acciones. En el caso específico de los operadores es necesario definir su precedencia y si son asociativos a la derecha o a la izquierda. Mediante estas definiciones se instruye al generador del analizador sintáctico en qué orden realizar las operaciones. En la Figura II.6 se muestra cómo fue definida la precedencia de los operadores en el sistema.

```
precedence right ASIGN;
precedence left LT, LE, GT, GE, EQ, NE;
precedence left MINUS, PLUS;
precedence left OR_T;
precedence left MUL, RDIV, DIV_T, MOD_T, AND_T;
precedence left NOT;
precedence right UMINUS;
precedence right POT;
precedence right FACT;
```

Figura II.4 Precedencia de los operadores

El orden de precedencia se declara desde abajo hacia arriba, como se muestra en el ejemplo, el operador de más precedencia es el factorial y el de menos la asignación.

II.4.2.3.2 Tratamiento de los errores

La herramienta CUP permite definir rutinas de manipulación de error en Java. Haciendo uso de esta funcionalidad se definieron los métodos report_error y report_fatal_error los cuales capturan los errores sintácticos y se le informa al usuario de los mismos mostrándole la línea en la que ocurrió. Aunque en el análisis sintáctico se realizan algunos chequeos semánticos, la gran mayoría de estos se analizan durante la inferencia, estos análisis se ilustrarán más adelante.

II.4.2.3.3 La Forma Interna

Para generar la forma interna fue necesario el diseño de un conjunto de clases que permitieran a la máquina de inferencia ejecutar las acciones descritas por el ingeniero del conocimiento en la base.

Para cada símbolo no terminal de la gramática se creó una clase asociada que tiene como parámetros objetos del tipo de los no terminales que se encuentran en esa derivación. Cada clase correspondiente tiene el mismo nombre que el símbolo no terminal solo que precedido de una T.

En la Figura II.7 se puede ver un ejemplo. En la misma se muestra la sintaxis de JFlex, cada producción puede o no tener acciones semánticas asociadas, estas no son más que código en Java y están delimitadas por **{::}**. En la Figura II.7 el no terminal "*value*" (en español valor), el cual puede ser una variable ("*var*") o una constante

("*const*"). En caso de que sea la primera se crea un objeto de tipo *Tvalue* con *var* como atributo, en el segundo caso se crea un objeto de tipo *Tvalue* con *const* como atributo. A su vez *const* puede ser una cadena (STRING_T), un número entero (INTEG), un número real (REAL_T) o un valor desconocido (_UNKNOWN). Todos estos valores son no terminales, por lo cual cuando se construye el objeto de tipo *Tconst* se le pasa por parámetros el valor que tiene el símbolo y no un objeto con el nombre del mismo, se le pasará el valor del número o la cadena que representa, o la cadena "UNKNOWN" en caso de que sea un valor desconocido. Lo mismo ocurre con *var*, solo que en este caso específico en lugar de pasarle el nombre de la variable únicamente, se le pasará también el código de esta en la tabla de símbolos, esto se hace llamando al método *getCodByName (int código)* de la clase *Symtab* (fue definida para representar la tabla se símbolos).

```
value   ::= var:v    {: RESULT = new Tvalue(v); :}
        | const:c    {: RESULT = new Tvalue(c); :}
        ;

var     ::= IDENT:id {: RESULT = new Tvar(id,this.tabla.getCodByName(id));:}
        ;

const   ::= STRING_T:t {:RESULT = new Tconst(t);:}
        |   INTEG:i {:RESULT = new Tconst(i);:}
        |   REAL_T:r {:RESULT = new Tconst(r);:}
        |   _UNKNOWN:_u {:RESULT = new Tconst("UNKNOWN");:}
        ;
```

Figura II.5 Segmento de la gramática representada usando CUP

Como se ha visto cada vez que en el análisis sintáctico se infiere por un no terminal se crea una clase con su nombre pasándole como atributos los objetos del tipo de las sentencias involucradas en la expresión. Como se puede inferir, la forma interna se representa con un objeto del tipo del símbolo no terminal por donde se comienza a analizar la gramática. El nombre del no terminal es *punto_partida* por tanto la clase que contiene toda la información concerniente a la forma interna se denomina *Tpunto_partida*. La estructura de la clase de define en la Figura II.8. En esta, var_list representa la lista de variables externas al sistema, rule_list la lista de reglas, encabezamiento contiene las acciones y asks_list los atributos que han de ser instanciados por el usuario durante la inferencia.

Figura II-6 Clase que representa la forma interna

La herramienta CUP tiene una sección donde permite definir código en Java. Esta es útil para definir métodos u objetos que se quieran utilizar en las acciones semánticas o luego de terminar el análisis sintáctico. En el caso del sistema esta sección fue utilizada para definir el método *writeFI*. A este se le pasa, por parámetros, una lista de objetos (usualmente un objeto de la clase Tpunto_partida y la tabla de símbolos) y el nombre del archivo donde se va a escribir, si el archivo existe se sobrescribirá y en caso contrario se creará. La forma interna se escribe cuando el analizador sintáctico termina su análisis.

II.4.3 La máquina de Inferencia

La máquina de inferencia es la encargada de ejecutar las acciones definidas por el usuario en la base de conocimiento y mostrar los resultados. Para poder iniciar el proceso de inferencia es necesario tener el archivo con la forma interna generada.

Durante la inferencia, un grupo de variables se modifican continuamente, ellas determinan el estado del proceso y todas se declaran como parámetros estáticos de la clase Inference, esto permite que sean accedidas sin la necesidad de crear un objeto específico de la clase. Muchas de estas son instanciadas al inicio de la inferencia. Los atributos más trascendentes son los siguientes:

- ***externalVars:*** Contiene una lista enlazada de Tvar. Al comienzo de la inferencia, recibe las variables externas que contiene la forma interna.
- ***rules:*** La lista enlazada de Trules se instancia con la lista de reglas de la base de conocimiento.
- ***asks:*** Las variables que serán instanciadas por el usuario durante la inferencia.

- ***facts:*** Contiene una lista enlazada de los hechos que ya han sido probados. Al inicio de la inferencia esta lista estará usualmente vacía, a menos que una de las variables que fue declarada como externa en la base de conocimiento actual haya sido instanciada con un valor en una inferencia previa (esto se comprende mejor más adelante cuando se explique la acción CHAIN).
- ***actions:*** Una lista enlazada con las acciones definidas en la base de conocimiento actual.
- ***table:*** La tabla de símbolos.

Lo más importante de la máquina de inferencia es la forma en que realiza la búsqueda y por consiguiente cómo son instanciados las variables de las cuales el usuario desea conocer su valor. A continuación se describe la forma en la que UCShell 2.0 realiza la inferencia.

II.4.3.1 Búsqueda en Backward

La búsqueda en backward o encadenamiento hacia atrás es una búsqueda dirigida por objetivos. En lugar de comenzar a probar qué resultado se alcanza con los datos que se tienen, como podría indicar la lógica humana, se busca directamente dónde está el objetivo y se analiza la vía mediante la cual se puede inferir. En caso de que las condiciones para llegar al objetivo no sean hechos probados, se procede a realizar la búsqueda de cada objetivo.

En el caso de UCShell 2.0 la acción que provoca la inferencia en backward se denomina FIND y su sintaxis es: FIND *varName*, donde *varName* es el nombre de la variable que se desea inferir. Cuando se ejecuta la acción FIND la máquina de inferencia realiza los pasos que muestra el pseudocódigo de la Figura II.9.

```
Recorre las reglas
        Toma la regla actual
        Si la variable está en la conclusión de la regla
            Prueba las condiciones de la regla
            Si se cumplen
                Devuelve el valor
                Termina
        Toma la próxima regla
```

Figura II.7 Pseudocódigo que representa el método FIND

En caso de que se ejecute una sentencia FIND y el sistema no encuentre ningún valor para la variable se retornará que el mismo es UNKNOWN o sea desconocido. Las razones por las cuales puede pasar que no se pueda instanciar una variable y el sistema devuelva que su valor es desconocido, pueden ser las siguientes:

- La variable no está asociada a ninguno de los hechos actuales del sistema.
- La variable no está dentro de la lista de *preguntables.*
- No se encuentra como conclusión de ninguna regla.
- No se pudieron probar las condiciones en las cuales la variable está como conclusión.

Un error común en el que puede incurrir el Ingeniero del Conocimiento consiste en intentar buscar el valor de una variable que no se encuentra en la lista de variables externas, no es un *preguntable* y no se encuentra en la conclusión de una regla, en ese caso el resultado de la inferencia será que la variable toma el valor UNKNOWN con un valor de certidumbre de 1.

II.4.3.1.1 Búsqueda de soluciones alternativas

En ciertas ocasiones puede ser necesario no solo tener un valor sino varios. Esto se ve usualmente en el campo de la medicina, donde el paciente puede no quedar muy seguro del diagnóstico y pedir al doctor que le dé una lista de enfermedades alternativas a la diagnosticada por el médico.

UCShell 2.0 resuelve este problema añadiendo la sentencia **FINDALL**. La ejecución de esta sentencia es similar a la de **FIND**, solo que esta no se detiene una vez instanciado el valor sino que sigue hasta haber recorrido todas las reglas. Al finalizar devolverá todos los valores con los que pudo instanciar a la variable y la certidumbre asociada a cada uno de ellos. Cada vez que encuentra un nuevo valor para la variable inserta un nuevo hecho en la lista de hechos de la clase *Inference*, de ahí que cuando termine de ejecutarse esta acción, en dicha lista puede haber más de una entrada con la misma variable. Esto lo podrá ver el usuario mediante la interfaz visual.

II.4.3.2 Búsqueda en Forward

A este tipo de búsqueda se le denomina también encadenamiento hacia adelante y búsqueda dirigida por datos. En esta, en lugar de buscar el objetivo en la conclusión de

las reglas, lo que se hace es ir recorriéndolas e ir probando con los datos que se tienen si se pueden arribar a las conclusiones. Esta es una búsqueda a ciegas porque no se sabe dentro de qué regla se podrá encontrar como conclusión la variable que se pretende instanciar. Incluso, puede que esta no se encuentre como conclusión de ninguna regla y el sistema no lo notará hasta haberlas recorrido todas.

Las bases de conocimiento se diseñan teniendo en cuenta el tipo de búsqueda mediante la cual se realizará la inferencia. Aunque con el método **FIND** se puede inferir en una base diseñada para hacer encadenamiento hacia adelante y viceversa, estos procesos son más óptimos cuando se utiliza el operador adecuado al diseño de la base.

II.4.3.3 Descomposición del sistema

UCShell 2.0 permite realizar la inferencia en múltiples bases de conocimiento. Gracias a esta facilidad se pueden agregar nuevas reglas al sistema sin que esto afecte considerablemente el tiempo de ejecución de la inferencia. La sentencia que hace posible esto es **CHAIN**.

La sintaxis de esta sentencia en **CHAIN** *fileName*, donde *fileName* es el nombre de la base de conocimiento (compilada o no) en la cual se quiere comenzar a inferir. En caso de que sea un archivo con extensión ".kbo" el sistema cargará la forma interna representada en el mismo. Además se puede poner también el nombre de la base de conocimiento sin compilar, en cuyo caso el sistema la compilará y comenzará la inferencia a partir de la forma interna creada.

La invocación de la sentencia provocará la actualización de las variables de la clase Inference. Esto ocurrirá de la siguiente forma:

- ***rulesList*** tomará ahora la lista de reglas de la nueva forma interna.
- ***asksList*** toma la lista de asks de la forma interna.
- ***varList*** toma la lista de variables que han sido declaradas externas en la nueva base de conocimiento. En el caso de que en la base anterior existan hechos asociados a las mismas se añadirán a la nueva base de conocimiento. En las versiones anteriores del software era necesario guardar los hechos mediante la acción **SAVEFACT** y cargarlos a través de **LOADFACTS**, estas acciones quedan obsoletas ya que los valores se salvan

automáticamente con el **CHAIN**, facilitando así el trabajo el ingeniero del conocimiento.

- ***encabezamiento*** toma la lista de acciones de la forma interna y una vez cargados el resto de los datos comienza a ejecutar las acciones. En caso de que en una base de conocimiento haya sido definida la acción CHAIN, tendrá que ser la última, teniendo en cuenta que la misma provoca que se cargue la forma interna de la base de conocimientos que se le indica, la cual incluye las nuevas acciones. Por tanto, una acción declarada luego de CHAIN en una misma base de conocimiento nunca será alcanzada.

II.4.4 Tratamiento de los errores

En el proceso de inferencia el sistema reconoce errores de tipo semántico. Cuando se intenta realizar una operación y los tipos de los operadores no son los permitidos se mostrará un mensaje de error al usuario. Estas situaciones no se pueden reconocer durante la compilación teniendo en cuenta que las variables toman el tipo del último valor que le fue asignado, de ahí que solo se pueda saber que hay error cuando se está a punto de realizar la operación. Cada operador tiene una clase asociada en la cual está implementada la rutina de manipulación de los errores a los que es susceptible.

El usuario del sistema puede ser fuente de errores también, en el caso de las preguntas continuas, al entrar tipos de datos incorrectos.

El mecanismo de inferencia fue implementado de tal forma que ignora estas situaciones y sigue adelante. Esto trae ventajas y desventajas: entre las ventajas esta el hecho de que el usuario puede ver varios errores en una misma inferencia y como desventaja que pueden aparecer varios errores cuando en realidad es uno solo y los demás tan solo son consecuencias del primero. No obstante, todos los errores reconocidos por el sistema serán mostrados al usuario una vez terminada la inferencia.

II.4.5 Incertidumbre

En ocasiones los datos con los que se cuenta no son precisos y se hace necesario asignarle distintos niveles de precisión. Este tipo de problemas se denominan estocásticos y UCShell 2.0 permite resolverlos.

Las causas de la incertidumbre pueden ser variables. El sistema da la posibilidad al usuario de definir la incertidumbre a los actores que usan el sistema de la siguiente manera:

Usuario: Cuando le da valor a un **ASK**.

Ingeniero del Conocimiento: Cuando diseña la gramática puede decidir que una regla es débil y asignarle un valor de certidumbre.

Los valores de certidumbre se mueven en el rango de 0 a 1, donde 1 indica que se está totalmente seguro, o sea, que no hay incertidumbre. El factor de certidumbre (FC) de las expresiones se calcula de la siguiente forma:

- **Para las condiciones A <operardor> B**, donde operador se refiere a los operadores binarios que permite la gramática, diferentes de AND y OR:

  FC(A <operador> B)=min (FC(A), FC (B)) *I.2*

- **Para la condición NOT(A):**

  FC (NOT A)= 1-FC(A) *I.3*

- **Condiciones unidas por la conectiva AND**:

  FC(A and B) = mín(FC(A), FC(B) *I.4*

- **Condiciones unidas por la conectiva OR:**

  FC(A or B) = máx(FC(A), FC(B)) *I.5*

- **Para el cálculo del FC de la conclusión de una regla:**
  Dada la regla: **IF** A **THEN** B **CNF** v:

  *FC (B) = FC(A) * v,* siendo v FC de la regla. *I.6*

II.5 Conclusiones del capítulo

En el presente capítulo se han mostrado los distintos productos de software que componen el sistema. Se ha hecho una descripción de los casos de uso de cada uno de estos. Se mostraron las etapas de diseño, partiendo de la sintaxis de la base de conocimiento, siguiendo con el proceso de compilación y finalmente con el de inferencia, explicando además los mecanismos de cálculo de certidumbre en esta última.

Capítulo III UCShell 2.0: Visión Externa

III.1 Introducción

UCShell 2.0 es un sistema que permite la implementación de Sistemas Expertos. Los usuarios del sistema pueden usar cada uno de los módulos según sus necesidades, en este capítulo se describen esas partes desde una visión externa, además de mostrar sus componentes visuales.

Cada uno de los módulos se puede instalar por separado ya que, aunque las funcionalidades de unos pueden estar implícitas en otros, funcionan de manera independiente.

III.2 Compilador de Bases de Conocimientos

Es el módulo encargado de compilar las bases de conocimiento, esta operación se puede hacer de dos formas:

- **Compilar con información real:** de esto se encarga el Compilador de Bases de Conocimiento UCShell Compiler 2.0.
- **Compilar con información simbólica:** el módulo encargado de ejecutar esta tarea es UCShell Symbolic Compiler 2.0.

La sintaxis de las bases reconocidas por ambos módulo es la misma que fue descrita en el capítulo anterior. Cada uno posee una interfaz visual sencilla que facilita la interacción con el usuario.

III.2.1 Compilador de Bases de conocimiento UCShell Compiler 2.0

Como se mencionó anteriormente, este módulo es el encargado de compilar bases de conocimiento con información real. Una vez terminado este proceso se genera una forma interna, que puede usarla cualquiera de los módulos en que esté implementada esa acción.

III.2.1.1 Requisitos de instalación

Teniendo en cuenta que el software fue programado en Java se puede instalar independientemente del Sistema Operativo que esté corriendo en la máquina. No obstante, atendiendo a esto se hace imprescindible que se encuentre instalada la

máquina virtual de Java. Una vez comprobado este hecho es necesario copiar la carpeta UCShell Compiler 2.0, la misma contiene el archivo UCShell_Compiler_2.0.jar y un conjunto de bibliotecas necesarias para su ejecución que están contenidas en la carpeta "lib". En general los archivos que conforman el Compilador de Bases de Conocimiento ocupan, como mínimo, aproximadamente 5MB de memoria. Este espacio puede aumentar de acuerdo a la longitud y cantidad de las bases de conocimiento que se compilen, pero el crecimiento no será significativo.

III.2.1.2 Interfaz Visual

UCShell Compiler 2.0 cuenta con una interfaz visual sencilla que permite seleccionar la base de conocimiento, compilarla y generar un archivo que contiene la forma interna. La Figura III.1 muestra la interfaz.

Figura III.1 Interfaz visual de UCShell Compiler 2.0

Para compilar un fichero primero es necesario seleccionarlo, a través del botón "Escoger Fichero", la acción clic mostrará un cuadro de diálogo para escoger el fichero, debe ser una base de conocimiento con extensión "*.kbs*". Dicho fichero puede encontrarse en cualquier parte del árbol de directorios y no necesariamente en el mismo donde está instalado el software. Una vez seleccionado el archivo se mostrará su dirección completa en el cuadro de texto y se activará el botón "Compilar" que inicialmente se encontraba desactivado.

Teniendo activado el botón "Compilar" se puede proceder a iniciar la compilación haciendo clic en el mismo. Una vez hecho esto el sistema iniciará los análisis

lexicográficos, sintácticos y semánticos según fueron definidos. Esta ejecución se puede terminar por dos causas:

- **Se terminó la compilación correctamente:** En este caso se creará un archivo, con la forma interna, que estará contenido en la misma localización que la base de conocimiento. El nombre del archivo será igual que el de la base de conocimiento pero con extensión ".kbo".

- **Se encontraron errores durante la compilación:** Se mostrará un mensaje diciendo que hubo errores y la dirección del archivo que contiene la descripción de los mismos. Usualmente este archivo se crea en el directorio donde se encuentra la base de conocimiento, el mismo contiene el número de línea donde se detectó el error. Es recomendable que el usuario, al tratar de enmendar los errores en la base de conocimiento, arregle uno a la vez, o sea, que enmiende el primer error y luego compile nuevamente la base de conocimiento en busca de más. Esto se recomienda ya que usualmente los errores se propagan y pueden aparecer varios como resultado de esa propagación.

El archivo que se genera y que contiene la forma interna puede ser utilizado por cualquiera de los módulos que realizan inferencia con información real.

III.2.2 Compilador de Bases de conocimiento con información simbólica UCShell Symbolic Compiler 2.0

Este módulo compila la base de conocimientos con información simbólica. Al terminar genera la forma interna. Esta es utilizada por productos de software tales como TeachShell para, a partir de la misma, realizar inferencias. **UCShell Symbolic Compiler 2.0** genera una forma interna diferente a la que generaría UCShell Compiler 2.0 sobre una misma base de conocimientos. De ahí que esta no será reconocida por los módulos que realizan inferencias con información real.

III.2.2.1 Requisitos de instalación

Además de disponer de la máquina virtual de Java, debe copiarse la carpeta UCShell Symbolic Compiler 2.0, el cual contiene el archivo UCShell_Symbolic_Compiler_2.0.jar y un conjunto de bibliotecas necesarias para su ejecución. En general los archivos que conforman el Compilador de Bases de Conocimiento con Información Simbólica ocupan como mínimo aproximadamente

4MB de memoria, a lo que se adiciona la longitud de las bases de conocimiento que se compilen.

III.2.2.2 Interfaz Visual

La interfaz visual de UCShell Symbolic Compiler 2.0 es muy similar a la de UCShell Compiler 2.0. La misma permite cargar una base de conocimiento y compilarla generando una forma interna de la misma forma que con información simbólica. Esta se muestra en la Figura III.2.

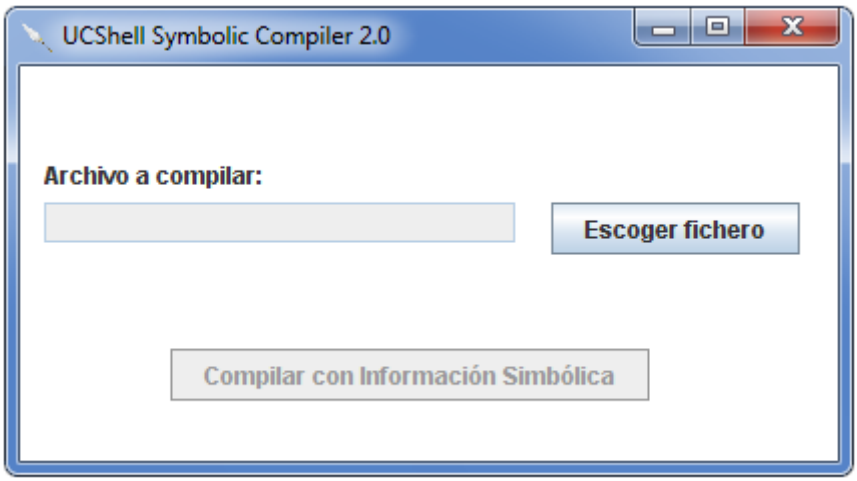

Figura III.1 Interfaz visual de UCShell Symolic Compiler 2.0

La base se carga exactamente igual que en UCShell Compiler 2.0. Una vez escogida se podrá pasar a compilarla dando clic en el botón "Compilar con Información Simbólica". Cuando se compila correctamente se muestra un mensaje al usuario indicando el nombre del archivo que contiene la forma interna.

III.2.2.3 Detección de errores

Los errores detectados se muestran en el archivo errors.txt, que se crea en el mismo directorio donde reside la base de conocimiento. A diferencia de UCShell Compiler 2.0 solamente son detectados los errores lexicográficos y sintácticos, los semánticos son ignorados.

Si se desea hacer un análisis más potente de la gramática se recomienda compilarla primero con UCShell Compiler 2.0 o con UCShell 2.0, una vez que se esté seguro de que no hay errores pasar a compilarla con UCShell Symblic Compiler 2.0 para generar la forma interna con información simbólica.

III.3 Biblioteca con mecanismos de inferencia y compilación UCShell Library 2.0

Los Sistemas Expertos se pueden aplicar en áreas del saber muy disímiles y la naturaleza de los datos que contienen puede ser muy variada, de ahí que de un sistema a otro varíe mucho la forma óptima de interactuar visualmente con el usuario. Los módulos anteriores fueron diseñados para mantener siempre la misma interfaz por tanto se hace necesario crear un sistema que permita modificar la interfaz de acuerdo a las particularidades del sistema que se quiere modelar y a la vez permita incorporar el mecanismo de inferencia de UCShell a otras aplicaciones .

UCShell Library 2.0 es una biblioteca que permite cumplir este objetivo. La misma contiene, entre otras funcionalidades, un compilador de bases de conocimiento y una máquina de inferencia. A continuación se hará una descripción del uso de la misma.

III.3.1 Requisitos para la utilización de UCShell Library 2.0

Cuando se desee crear un proyecto que haga uso de UCShell Library 2.0, será necesario agregar, a las bibliotecas de dicho proyecto, los archivos "UCShell Library.jar" y "java-cup-11a.jar", el primero es la biblioteca propiamente dicha y el segundo se usa para la compilación.

UCShell Library 2.0, contiene mecanismos contiene mecanismo de compilación y de inferencia. Seguidamente se mostrará una guía para el desarrollo de sistemas usando esta biblioteca.

III.3.2 Compilación

UCShell Library 2.0 permite realizar la compilación con información real y con información simbólica. A continuación se explicará cómo realizar cada una de estas.

III.3.3 Compilación con información real

Para compilar una base de conocimiento hay que importar la clase "Compiler" que está dentro del paquete "compiler". La representación gráfica de la misma se muestra en la Figura III.3. A continuación se hará una descripción de los atributos y métodos de la clase.

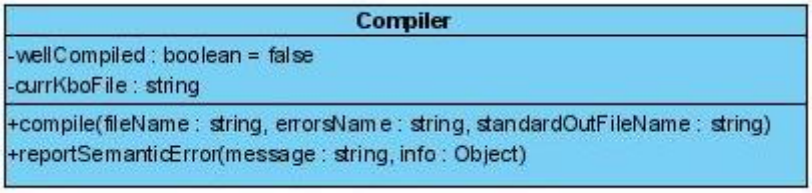

Figura III-2 Descripción de la clase Compiler

Atributos:

- ***boolean*** *wellCompiled:* Este atributo, por defecto, toma como valor falso. Una vez terminada la compilación y en caso de que haya sido exitosa, toma el valor *true* (verdadero).

- ***String*** *currKboFile*: Contiene el nombre del archivo generado luego de compilar, el que tiene la forma interna de la base de conocimientos.

Métodos

- ***void*** *compile:* Este es método más importante para el manejo de la compilación. Al mismo se le pasan, como atributos, el nombre del archivo a compilar (*filename*), el nombre del archivo donde se escribirán los errores (errorsName), y el de la salida estándar (*standardOutFileName*); este último archivo guardará los mensajes que envíe el sistema durante la compilación, usualmente estará vacío. Con estos parámetros, el método *compile* ejecuta la compilación. El archivo generado se guardará con el mismo nombre de la base de conocimientos (solo cambiará la extensión de ".kbs" por ".kbo") y en el mismo directorio. El resto de los archivos se pueden encontrar en cualquier lugar del árbol de directorios.

- ***void*** *reportSemanticError*: Este método eleva un error semántico, el sistema internamente lo llama cuando ocurre un error de este tipo.

Una vez compilada una base de conocimiento se puede inferir a partir de la forma interna generada. El modo de acceso a los métodos para realizar la inferencia se describe a continuación.

III.3.4 Compilación con información simbólica

Para compilar con información simbólica, hay que importar la clase "SymbolicCompiler" que se encuentra dentro del paquete "ts.comp". La sintaxis de esta clase es semejante a la de "Compiler". Ambas tienen los mismos atributos y métodos, las diferencias en ellas están dadas en la forma en que implementan estos últimos, por tanto se utilizan de la misma forma.

III.3.5 Inferencia

Para inferir hay que hacer uso del paquete "inference", este contiene la clase "Inference" que es la que implementa los métodos para inferir y además carga la forma interna de la base de conocimiento. En la Figura III.4 se puede observar una representación gráfica de esta clase con sus principales atributos y métodos. A continuación se hará una descripción de los mismos.

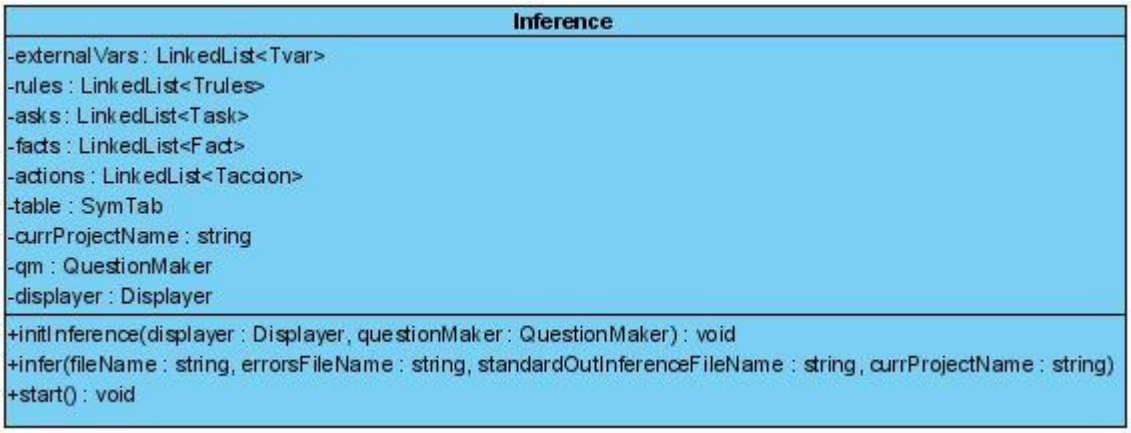

Figura III.3 Descripción de la clase Inference del paquete inference

Atributos

- Atributos que toman los valores de la forma interna:
    - ***LinkedList<Tvar>*** *externalVars*: Variables externas al sistema.
    - ***LinkedList<Trules>*** *rules*: Reglas de la Base de Conocimiento.
    - ***LinkedList<Tasks>*** *asks*: Atributos preguntables.
    - LinkedList<Taccion> actions: Acciones.
    - ***SymTab*** *table*: Tabla de símbolos.

- ***LinkedList<Tfact>*** *facts*: Los hechos toman valores durante la inferencia.

- ***String*** *currprojectName*: Nombre del proyecto actual.

- ***QuestionMaker*** *qm:* Objeto que implementa la interfaz QuestionMaker. Define el comportamiento de las ventanas con las cuales interactúa el usuario para responder preguntas y para dar valores de certidumbre. El método a implementar en esta interfaz es:

    - *Fact makeQuestion (Ask ask)*: En este método se define la forma en que se interrogará al usuario. Al mismo se le pasa por parámetros un objeto de tipo Ask. Este contiene toda la información relativa a la pregunta. Una vez contestada la misma, se devuelve un objeto de tipo Fact que contiene los detalles de la respuesta dada por el usuario.

- ***Displayer*** *displayer:* Objeto que implementa la interfaz Displayer. Define cómo mostrará los datos al usuario, ya sean texto, imágenes o video. Los métodos a implementar son:

    - ***void*** *display (**String** cadena)*: se define cómo el sistema mostrará una cadena.

    - ***void*** *displayWithImage (**String** imageDir)*: se define cómo el sistema mostrará una imagen, la dirección de la misma es *imageDir*.

    - ***void*** *displayWithVideo (**String** video)*: se define cómo el sistema mostrará un video. La ubicación del video está contenida en la cadena *video*.

Métodos

- ***void*** *initInference (Displayer displayer, QuestionMaker qm)*: A este método es necesario llamarlo antes de realizar la inferencia, el mismo le asigna a la clase *Inference* los objetos que implementan la interfaz *Displayer* y *QuestionMaker* respectivamente.

- ***boolean*** *infer (String filename, String errorsFileName, String standardOutInferenceFileName, String currProjectName)*: Este método es el que carga la forma interna y realiza la inferencia, en caso de que no haya

errores durante devolverá *true*, en caso contrario *false*. Los parámetros que se le pasan al método son:

- ***String*** *fileName**:*** El nombre del archivo que contiene la forma interna de la base de conocimiento.
- ***String*** *errorsFileName:* el nombre del archivo donde se guardarán los errores detectados durante la inferencia.
- ***String*** *standardOutInferenceFileName*: El nombre del archivo donde se guardarán los mensajes enviados por el sistema, en caso de que se envíe alguno.
- ***String*** *currProjectName*: La dirección del proyecto actual.

De manera general, cuando se quiere inferir se ha de crear un objeto de la clase Inference, luego llamar al método *initInference* pasándole por parámetros los objetos *displayer* y *questionMaker*. Finalmente se llama al método infer con los parámetros requeridos. Una vez terminada la inferencia los hechos probados se encontrarán en el atributo *facts* de la clase *Inference*.

III.4 Ambiente de desarrollo integrado

Dentro de los módulos de UCShell 2.0, se incluye un Ambiente de desarrollo integrado (IDE por sus siglas en inglés) llamado UCShell IDE 2.0. Este hace uso de la biblioteca UCShell Library 2.0 para realizar los mecanismos de inferencia y compilación. En el mismo se puede, además, editar las bases de conocimiento con las facilidades propias de este tipo de sistema tales como: resaltado y completamiento de las palabras reservadas, búsqueda de expresiones, copiado, pegado etcétera.

III.4.1 Instalación y requisitos

Para instalar este software es necesario tener previamente instalada la máquina virtual de Java con el jdk 1.5 o superior. Dentro de la carpeta UCShell IDE 2.0 se encuentran los archivos o carpetas necesarios para su correcto funcionamiento. En caso de borrarse alguno de estos el sistema presentará fallos. Ocupa un espacio aproximado de 6 MB que podrá aumentar de acuerdo al tamaño de las bases de conocimientos que se creen.

III.4.2 Interfaz gráfica de usuario

La interfaz gráfica de UCShell IDE 2.0 permite, no solo compilar y ejecutar bases de conocimientos, sino que también se pueden editar. El entorno gráfico se sirve de menús y cuadros de diálogo que facilitan la interacción con el conocimiento, ocultando los procesos que pueden ser de difícil comprensión para el usuario final, el cual usualmente no tiene conocimientos avanzados de computación. La ventana principal del sistema se puede ver en la Figura III.5. Puede apreciarse, a la izquierda, el árbol de proyectos y a la derecha el panel donde se carga la base de conocimiento seleccionada para su edición y debajo el panel donde se muestran los mensajes del sistema.

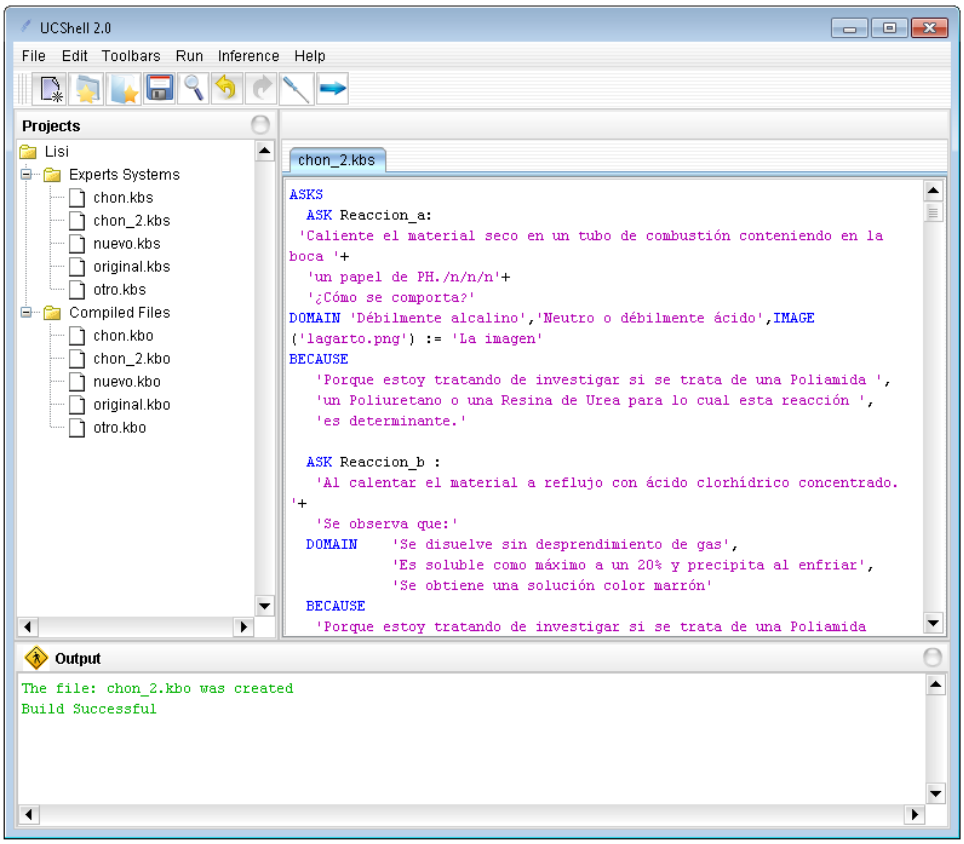

Figura III.4 Ventana principal de UCShell IDE 2.0

Se puede observar además la barra de herramientas, la cual ofrece un acceso rápido a algunas de las funcionalidades del software. Mediante la barra de Menú se puede acceder a la totalidad de estas. A continuación se hará una descripción de la misma.

III.4.2.1 Barra de Menú:

Se encuentra debajo de la barra de título, permite acceder a todas las funcionalidades del software. Está compuesta por varios sub-menús agrupados de acuerdo a sus funciones respectivas.

III.4.2.1.1 Menú File

Agrupan las tareas asociadas a los archivos y proyectos, las opciones que ofrece son:

New Project: Crea un nuevo proyecto. Para una mejor organización del contenido se recomienda que cuando se vaya a crear un sistema experto, se cree un nuevo proyecto y dentro del mismo se guarde toda la información relativa al mismo. Teniendo en cuenta que UCShell permite el uso de bases de conocimiento descomponibles, cuando se vaya a hacer uso de esta funcionalidad se recomienda que todas las bases de conocimiento que intervengan sean guardadas dentro del mismo proyecto, no obstante el sistema no obliga a esto. Una vez que se da clic en New Project el sistema muestra el cuadro de diálogo de la Figura III.6, que solicita el nombre del proyecto que se creará dentro de la carpeta ***workspace*** que se encuentra en el directorio donde está instalado el software. Se da la opción de crear además un archivo *.kbs*, o sea, una base de conocimiento nueva dentro del proyecto, la extensión (*.kbs*) no se debe cambiar, de lo contrario el sistema mostrará un mensaje de error. En caso de que el Sistema Experto que se va a crear trabaje con imágenes o video, el usuario ha de seleccionar la opción de establecer un directorio para los recursos. Este será creado dentro del proyecto actual con el nombre de "resources". En caso de que se trabaje con recursos y no se guarden dentro de esta carpeta puede haber fallos en la ejecución. Una vez creado exitosamente el proyecto, este se abrirá en el editor. Si se seleccionó la opción de crear también una base de conocimiento, esta tendrá inicialmente el código de la plantilla por defecto del sistema.

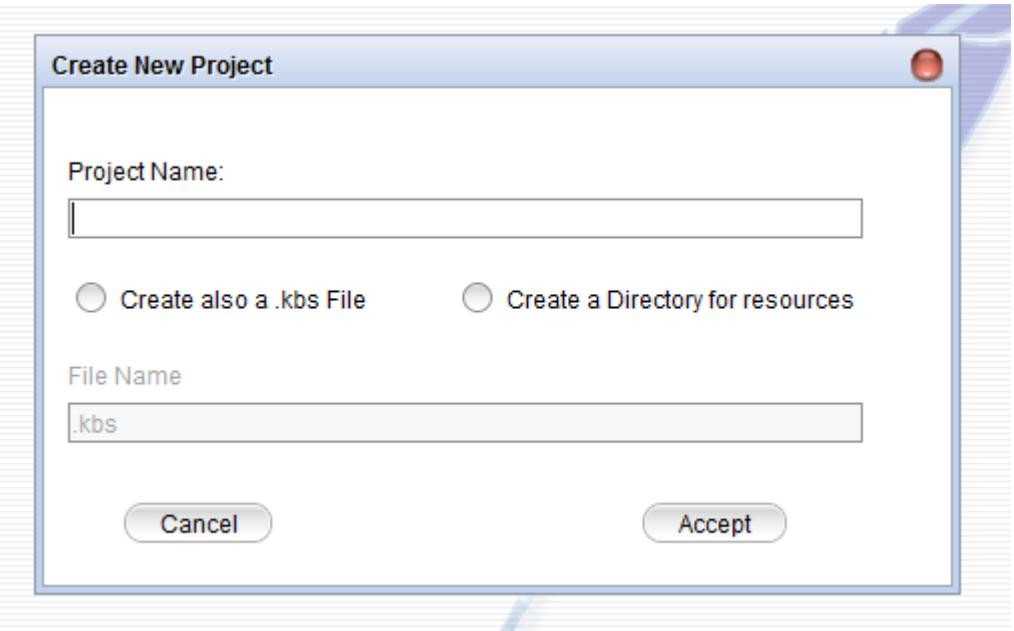

Figura III.5 Cuadro de diálogo para crear proyecto

New File: Muestra un cuadro de diálogo que pide el nombre de la base de conocimiento, la cual será creada dentro del proyecto actual, la extensión debe ser *.kbs*, de lo contrario el sistema mostrará un mensaje de error. Una vez creada exitosamente, tomará por defecto el valor de la plantilla definida en el sistema.

Save File: Salvará el archivo que el sistema tiene marcado como actual en caso de que su contenido haya variado, en cuyo caso se mostrará un asterisco en la pestaña correspondiente.

Open Project: Abre un proyecto existente y muestra su contenido en el árbol de la izquierda. En el caso de que tenga bases de conocimiento, las bases serán agrupadas dentro de la rama *Knowledge Bases* (Bases de Conocimiento), hay que mencionar que este no es un directorio real, sino que el sistema usa esta rama virtual para organizar la base. Las bases podrán ser seleccionadas y su contenido podrá editarse. Los archivos que contengan la forma interna de una base de conocimiento (los que tienen extensión *.kbo*) también se cargarán en el árbol, en este caso dentro de la rama *Compiled Files* (Archivos Compilados). Sin embargo, el contenido de esto archivos no podrá ser cargado ni editado ya que esto no tendría sentido porque su contenido es código objeto y no cadenas de caracteres.

III.4.2.1.2 Menú Edit

Permite editar el texto de la base de conocimiento. Sus opciones **Undo** y **Redo** las permiten deshacer y rehacer respectivamente los cambios realizados sobre el texto. Otra opción que brinda este menú es Find (Buscar).

Find: Permite buscar una palabra dentro del texto, para lo cual se usa el cuadro de diálogo de la Figura III.7. En caso de existir la palabra, cada una de sus apariciones será resaltada, en caso contrario mostrará un mensaje diciendo que no se encontró ninguna. La búsqueda que realiza es sensible a mayúsculas, o sea, que buscará la palabra escrita exactamente igual, solamente busca palabras completas. Otra opción que ofrece es la de reemplazar la palabra por otra.

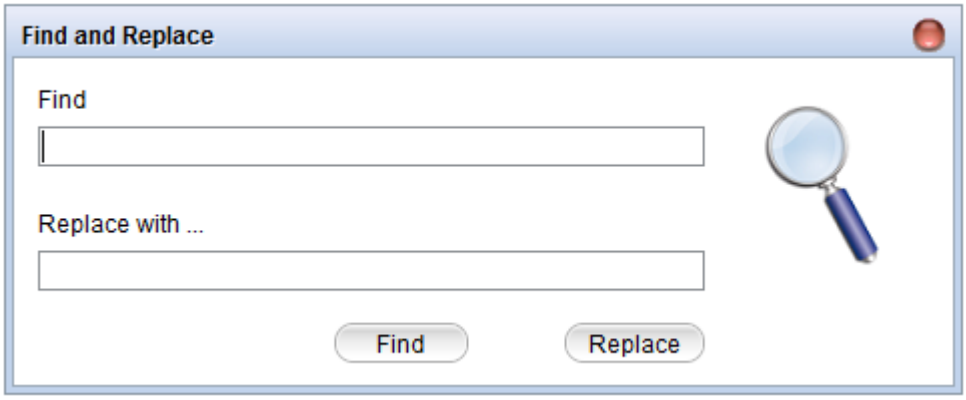

Figura III.6 Cuadro de diálogo para buscar y reemplazar

III.4.2.1.3 Menú Toolbars

Se pueden abrir vistas que fueron cerradas previamente. Las opciones son:

Output: Muestra el cuadro de salida, que se encuentra en la parte de debajo de la interfaz visual y contiene los mensajes que envía el sistema, incluyendo los de error.

All Projects: Muestra el árbol de proyectos en caso de que este haya sido cerrado anteriormente. Se encuentra a la izquierda en la interfaz visual y es el que contiene el proyecto actual con los archivos que contiene.

III.4.2.1.4 Menú Run

Este es quizás el Menú más importante, ya que contiene los principales métodos para compilar e inferir. Para que se puedan ejecutar cada uno de estos es necesario que se encuentre seleccionado un archivo.

Compile (Compilar): Toma la base de conocimiento seleccionada y la compila. Si se compila correctamente creará un archivo con la forma interna generada a partir de la compilación, además mostrará un mensaje en el panel de salida diciendo que se creó con éxito dicho archivo y los muestra en el árbol de la izquierda. En caso contrario mostrará los errores en el panel de salida y el número de línea en la que aparecen.

Compile Symbolic Information (Compilar con información simbólica): Compila con información simbólica. Este tipo de compilación no reconoce los comentarios, de ahí que en caso de que existan habrá que eliminarlos o el sistema dará un error cada vez que encuentre uno.

Run (Ejecutar): Este botón es el que ejecuta la inferencia. En caso de que se intente inferir a partir de una base de conocimiento que no ha sido previamente compilada el sistema mostrará, en el panel de salida, que hubo un error de entrada-salida. Si se encuentra algún error mientras se infiere se detendrá la inferencia y se mostrará el mensaje de error asociado. Si no hay ninguno de estos inconvenientes se comienza la inferencia. Durante la misma el sistema va interactuando con el usuario haciéndole preguntas.

Tipos de preguntas de acuerdo al contenido:

- No contienen imágenes: En este caso se muestra solo el texto asociado a ellas.

- Contienen imágenes: Si la imagen está asociada a la pregunta en sí entonces en el cuadro donde debe aparecer esta aparecerá la imagen. Si existen opciones que tengan asociadas imágenes, cuando se seleccione una de estas, a la derecha se mostrará la vista previa de la imagen. Si se quiere observar mejor se puede dar doble clic sobre ella para ampliarla.

Tipos de preguntas de acuerdo a las opciones:

- Son discretas: Existe una cantidad finita de posibles respuestas, cada una de estas opciones se muestra y el usuario las puede seleccionar.

- Son continuas: En este caso se muestra un cuadro de diálogo donde el usuario introducirá el valor que estime conveniente. Este tipo de respuestas es susceptible a error ya que puede ser que el sistema espere un número y el

usuario entre una cadena o viceversa. En caso de ocurrir se mostrará en el panel de salida el error.

Otro valor que puede controlar el usuario es la certidumbre con que da su respuesta, por defecto es 1. En caso de que el usuario quiera modificarla activará el botón **Certainly** (certidumbre) e introducirá el valor de certidumbre de su respuesta que ha de oscilar entre 0 y 1. Si se introduce un valor incorrecto el sistema enviará un mensaje de error.

En ciertas ocasiones un usuario puede tener dudas acerca del **porqué de la pregunta**, de ahí que cuando el ingeniero del conocimiento diseña la base puede incluirle a cada pregunta la razón por la cual es necesario hacerla. En el caso de que la pregunta tenga una explicación, esta podrá ser consultada por el usuario dando clic en el botón Because (por qué), el cual se encontrará activado si la explicación existe. En caso contrario se encontrará desactivado.

En la Figura III.8 se puede ver un ejemplo de pregunta hecha por el sistema al usuario. En la misma no hay imágenes asociadas y el dominio es discreto ya que existen opciones de respuesta. Además no existe una explicación asociada ya que el botón *Because* se encuentra desactivado.

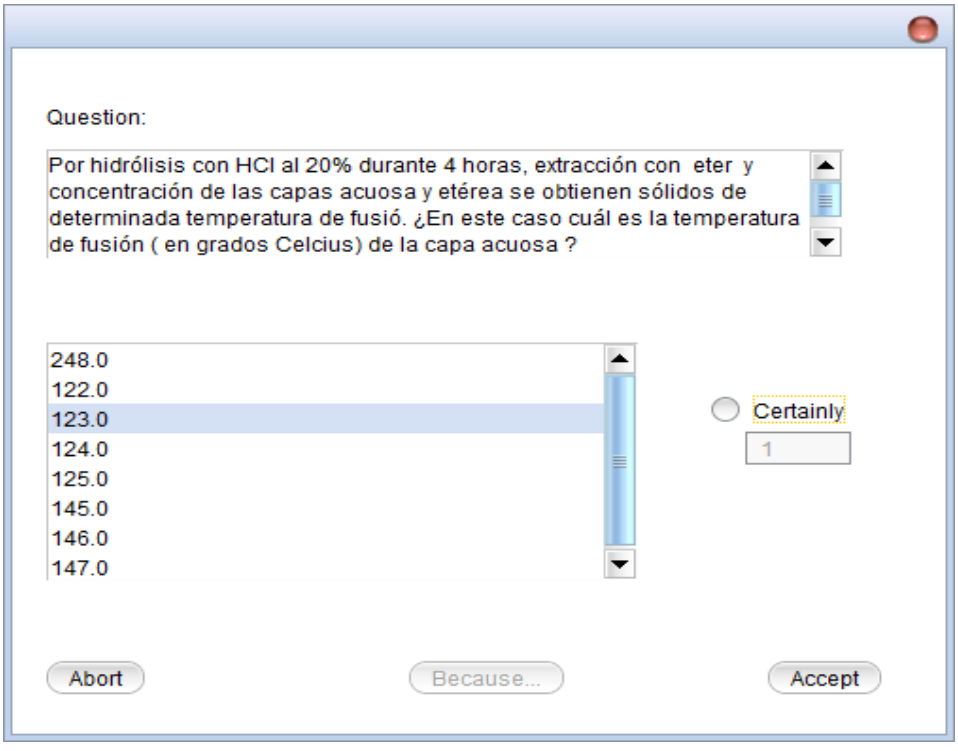

Figura III.7 Ejemplo de pregunta hecha por el sistema

III.4.2.1.5 Menú Inference

Haciendo uso de este menú se pueden tener detalles acerca de la última inferencia realizada por el sistema.

View Facts (Ver Hechos): Esta opción permite ver los hechos que fueron probados durante la inferencia. Solo está disponible en el caso que se haya inferido previamente. Cuando se selecciona, el sistema muestra un cuadro de diálogo donde, de cada hecho, muestra el nombre de la variable, el valor y el factor de certidumbre con el cual fue alcanzado. En la Figura III.9 se muestra un ejemplo de esta vista.

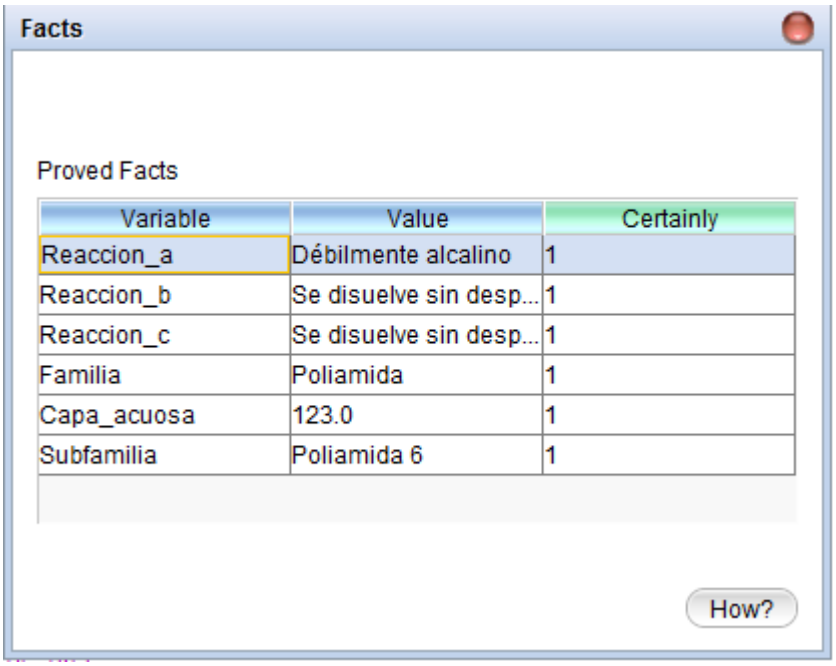

Figura III.8 Ejemplo de hechos probados luego de la inferencia

Los **hechos se muestran** de acuerdo a su orden de creación. No obstante se pueden ordenar luego según estime conveniente el usuario, de acuerdo al valor de cada una de las columnas, para hacer esto basta con dar clic en la columna que se quiere establecer como guía y el sistema la ordenará partiendo de los valores. Si se quiere cambiar de ascendente a descendente o viceversa basta con volver a dar clic sobre la columna.

Si se quiere saber **cómo** el sistema logró instanciar una variable con sus valores asociados en la tabla, se selecciona la fila, inmediatamente se activará el botón *How* (Cómo) que inicialmente estaba desactivado, cuando se da clic en ese botón, el sistema mostrará, en forma de texto, el camino que recorrió para instanciarlo.

III.4.2.1.6 Menú Help

Muestra la ayuda del sistema. En el mismo aparecen además referencias a la creación del sistema y a sus desarrolladores.

About: Muestra un cuadro de diálogo donde aparece información acerca del producto. Trae una panorámica muy general del uso del sistema y dónde y por quiénes fue implementado. Además aporta al usuario también la información de contacto de los desarrolladores mediante la cual pueden consultar dudas o dar sugerencias.

III.5 Conclusiones del capítulo

En el presente capítulo se ha mostrado detalladamente cómo acceder a las funcionalidades brindadas por cada uno de los productos de software que conforman UCShell 2.0. Además se han expuesto sus propiedades y requisitos de instalación.

Conclusiones

En este trabajo se logró crear una nueva versión de UCShell, esta vez con sus funcionalidades distribuidas en varios módulos. Se mejoraron los mecanismos de inferencia mediante la introducción de dos nuevas sentencias al lenguaje: FINDFORWARD y FINDALL. Además se simplificó el trabajo del ingeniero del conocimiento al eliminar sentencias que podrían nublar el diseño de la base, permitir la introducción de comentarios dentro de la misma y ampliar el reconocimiento de las expresiones. Cada uno de los módulos del sistema se implementó en Java y cuenta con un manual de usuario que facilita su uso. Se creó además un manual del programador con vista a realizar futuras modificaciones a estos productos de software.

RECOMENDACIONES

Se recomienda establecer UCShell 2.0, como software para la enseñanza de Sistemas Expertos dentro de la asignatura de Inteligencia Artificial en la UCLV, en sustitución de su versión anterior. Además se exhorta a que la utilización del mismo no se limite a la educación, sino que se extienda a resolver problemas reales presentes en nuestra sociedad.

Para aumentar las funcionalidades de UCShell IDE 2.0 se propone la inclusión al mismo de mecanismos de depuración de código.

Bibliografía

Alfred V. Aho, J. D. U. (2006). Compiladores: principios, técnicas y herramientas., Addison Wesley.

Bello, R. (2002). Aplicaciones de la Inteligencia Artificial.

Bratko, I. (1990). Prolog: programming for artificial intelligence, Addison Wesley.

Edward Feigenbaum, P. M. (1983). The fifth generation, Addison Wesley.

George F. Luger and W. A. Stubblefield (1998). Artificial Intelligence. Structures and Strategies for Complex Problem Solving.

Gries, D. (1971). Construcción de Compiladores.

Gutiérrez, J. M. (1998). Modelos de redes probabilísticos para Sistemas Expertos. V Conferencia Nacional de Ciencia de la Computación. Universidad de Cantabria.

Lezcano, M. (1998). Ambientes de aprendizaje por descibrimiento para la asignatura de Inteligencia Artificial. Inteligencia Artificial. Santa Clara, Universidad Central "Marta Abreu" de LAs Villas. **Doctor en ciencias**.

Lezcano, M. (2000). Prolog y los Sistemas Expertos, Jalisco.

Morales, Y. G. (2011). Implementación del software para la creación de Sistemas Expertos. Informática Educativa. Santa Clara, Universidad Central "Marta Abreu" de LAs Villas.

Norvig, S. R. a. P. (2004). Inteligencia Artificial, un enfoque moderno. Madrid, Prentice Hall.

Partridge, D. (1996). Artificial Intelligence, Academic Press.

Raynor, W. J. (1999). The International Dictionary of Artificial Intelligence. New York, Glenlake Publishing Company.

Rossel, G. (2010). Fundamentos de la inteligencia computacional.

Ruth Aylett, C. D. (2002). Supporting the domain expert in planning domain construction.

Sergio Gálvez and M. A. M. Mata (2005). Compiladores, Universidad de Málaga.