CAPITULO IV

Sistemas Expertos Basados en Reglas

1. Introducción

La Inteligencia Artificial (IA) es la parte de la Ciencia que se ocupa del diseño de sistemas de computación inteligentes, es decir, sistemas que exhiben las características que se asocian a la inteligencia en el comportamiento humano, esto es: la comprensión del lenguaje, el aprendizaje, el razonamiento, la resolución de problemas, etc [6]. Hoy en día, la inteligencia artificial abarca varias sub-áreas tales como los sistemas expertos, la demostración automática de teoremas, el juego automático, el reconocimiento de la voz y de patrones, el procesamiento del lenguaje natural, la visión artificial, la robótica, las redes neuronales, etc.

El presente capítulo tiene como objetivo introducir los conceptos referidos a sistemas expertos, y en particular sistemas expertos basados en reglas. Este último tipo de sistemas fue el seleccionado para implementar la propuesta de tesis que se detallará en el siguiente capítulo.

2. Sistemas Expertos

El diccionario de la Real Academia Española [7] define a este tipo de sistemas de la siguiente manera: “Sistema Experto: Programa de ordenador o computadora que tiene capacidad para dar respuestas semejantes a las que daría un experto en la materia.”

Los sistemas expertos surgen con la intención de emular el comportamiento de una persona experta en algún dominio. Los seres humanos llegan a convertirse en expertos

mediante el estudio y la práctica continuada que realizan sobre determinada materia. El conocimiento de un experto es una suerte de miscelánea entre los conceptos teóricos adquiridos y un conjunto de reglas heurísticas sobre un dominio, que la experiencia tanto propia como colectiva han establecido como válidas y efectivas [8].

Luego de un tiempo, una persona que en algún momento se la consideraba principiante en torno a un determinado dominio se transforma en un vasto conocedor del mismo, lo que suele llamarse, un experto en el tema. De la aplicación de este concepto a sistemas de software, surgen preguntas del estilo ¿Cómo es que un sistema llega a convertirse en un experto? ¿Estudia? ¿Practica?

A diferencia de los seres humanos, los sistemas expertos no se convierten en tales a partir del estudio y la practica constante. Los sistemas expertos no aprenden de su experiencia. El conocimiento humano de carácter experto sobre un dominio es traducido a un lenguaje formal con la finalidad de que las computadoras puedan hacer uso del mismo y sean capaces de generar respuestas a eventuales escenarios. Los sistemas expertos suelen estar focalizados en un reducido conjunto de problemas, es decir, su conocimiento es específico (al igual que los seres humanos, una persona no puede ser especialista en todas las materias). Generalmente, los sistemas expertos poseen las siguientes características [8]:

- Su estado mental, o base de conocimientos, es dinámico, es sencillo el agregado o substracción de conocimientos.
- Soportan y emulan el proceso de razonamiento humano mediante la manipulación de esta base de conocimientos.
- Razonan de forma heurística, usando su conocimiento para obtener la mejor solución al problema planteado.

2.1. Tipos de Sistema Experto

Los problemas con los que tratan los sistemas expertos pueden clasificarse en dos tipos: problemas esencialmente determinísticos y problemas esencialmente estocásticos [16]. Los problemas determinísticos son aquellos que pueden definir sus salidas si las entradas están definidas, y los estocásticos son aquellos en los que la solución no viene dada únicamente por los datos de entrada, sino que presentan un grado de incertidumbre y varían de acuerdo a alguna variable. Consecuentemente, los sistemas expertos pueden clasificarse en dos tipos principales según la forma en que implementan el conocimiento. Cada uno apunta a resolver problemas de distinta naturaleza [2].

- **Sistemas expertos basados en reglas**: son formulados utilizando un conjunto de reglas que relacionan varios objetos bien definidos. Los sistemas expertos de este tipo sacan sus conclusiones basándose en un conjunto de reglas utilizando un mecanismo de razonamiento lógico e intentan resolver problemas determinísticos [8].

- **Sistemas expertos probabilísticos**: en situaciones inciertas, es necesario introducir medios para tratar la incertidumbre. Estos sistemas utilizan la probabilidad como medida de incertidumbre. La estrategia de razonamiento que usan se conoce como razonamiento probabilísticos, o inferencia probabilística, suelen implementarse mediante redes bayesianas e intentan resolver problemas estocásticos [2].

Otro tipo de sistemas expertos son los combinados con el razonamiento basado en casos (CBR, Case Based Reasoring), en donde se cuenta con el razonamiento del experto y experiencia en casos previos para resolver los problemas. Los casos se encuentran almacenados en la base de datos del sistema experto y son consultados para resolver los diferentes problemas. Estos sistemas pueden usarse para resolver problemas de ambos tipos (determinísticos y estocásticos) [9].

2.2. Aplicación de Sistemas Expertos

Los sistemas expertos pueden ser aplicados en diversas áreas donde se necesite emular el razonamiento humano. Una de las ventajas principales de estos sistemas es que se pueden combinar conocimientos de varios expertos simulando el razonamiento conjunto de los mismos. Otra ventaja que presentan es que, estos sistemas, pueden utilizarse como fuente de conocimiento para realizar consultas del dominio sobre el cual se está resolviendo un problema.

El uso de sistemas expertos se recomienda especialmente en las siguientes situaciones [2]:

- Cuando el conocimiento es difícil de adquirir o se basa en reglas que sólo pueden ser aprendidas de la experiencia.
- Cuando la mejora continua del conocimiento es esencial y/o cuando el problema está sujeto a reglas o códigos cambiantes.
- Cuando los expertos humanos son caros o difíciles de encontrar.
- Cuando el conocimiento de los usuarios sobre el tema es limitado.

3. Sistema Basado en Reglas

Los programas basados en reglas surgen con la intención de ser usados para asistir en la resolución de problemas que con los métodos tradicionales de programación suelen ser difíciles de solucionar [8].

Es importante saber elegir el paradigma de programación adecuado para cada problema. Se puede dividir a los paradigmas en dos grandes grupos, aquellos en los que el programador codifica qué hacer y cómo hacerlo, y aquellos en los qué se codifica que hacer pero no se especifica de qué manera [8].

Dentro del primero grupo se encuentra la programación procedural y la programación orientada a objetos, entre otros. Aunque la estructura del programa difiera en los dos paradigmas, en ambos el programador codifica la lógica que controla a la computadora: el programador le indica a la computadora qué hacer, de qué manera, y en qué orden. Este enfoque se adapta perfectamente cuando las entradas se encuentran bien especificadas y se conoce un conjunto de pasos para poder alcanzar la solución al problema. Los cálculos matemáticos, por ejemplo, son exitosamente resueltos por este estilo de programación.

En el segundo grupo, se encuentran los denominados programas basados en reglas, los cuales poseen una naturaleza declarativa. Un programa puramente declarativo especifica qué debe hacer la computadora, sin embargo no especifica **cómo** debe hacerlo. Esta característica los hace generalmente más sencillos de comprender que los programas procedurales. Un programa declarativo no está compuesto por una larga secuencia de instrucciones, por el contrario, este se materializa en un conjunto discreto de reglas las cuales describen subconjuntos del problema [8].

Los programas declarativos deben ser ejecutados por un sistema que sea capaz de manipular información declarativa para solucionar los problemas. Debido a que el flujo de ejecución es controlado por este sistema, este tipo de programas son más aptos que otros para alcanzar una solución válida cuando el conjunto de datos de entrada se presenta pobre o incompleto. La programación declarativa es usualmente la manera natural de atacar problemas que involucran control, diagnóstico, predicción, clasificación, reconocimiento de patrones y cualquier situación problemática que no tenga asociada una solución algorítmica clara.

En resumen, en un programa basado en reglas, se escriben las reglas que describen el problema. Luego, otro programa, denominado motor de reglas, determina qué regla se aplica en un determinado momento y la ejecuta de manera adecuada. Como resultado, una versión basada en reglas de un programa complejo, generalmente, será más corta y simple

de entender que la versión procedural. Escribir el programa es sencillo, ya que el programador se puede concentrar en una situación a la vez y generar las reglas pertinentes.

3.1. Reglas y Motores de Reglas

Un sistema basado en reglas utiliza reglas para derivar conclusiones desde sus premisas.

Una regla es una especie de instrucción o comando que aplica en ciertas situaciones. En general, cualquier tipo de información que puede ser pensada en términos lógicos puede ser escrita en forma de regla. Las reglas se parecen mucho a las sentencias *if-then* de los lenguajes de programación tradicionales. La parte *if* de la regla, parte izquierda, es habitualmente conocida como predicados o premisas; y la parte *then*, parte derecha, como acciones o conclusiones. En ingles se las suele denominar como *LHS* (*Left-Hand Side*) a la parte izquierda y *RHS* (*Right-Hand Side*) a la parte derecha. Además, las reglas tienen asociado un dominio, el cuál es el conjunto de información con la que ésta puede trabajar.

Un sistema basado en reglas es aquel que utiliza un conjunto de reglas de inferencia para implementar el razonamiento de un experto. El conjunto de reglas que denotan el conocimiento son inyectadas dentro del motor de reglas, el cual tiene la capacidad de recordar, borrar o generar nuevas reglas.

Estos sistemas combinan la flexibilidad y eficiencia que provee un motor de reglas con la información experta obtenida sobre un dominio, pudiendo generar respuestas inmediatas y precisas a problemáticas particulares.

Algunos ejemplos de motores de inferencia utilizados para escribir sistemas expertos son: Dendral, XCon [13], Mycin [12], R1 [13], CLIPS [10] y Jess [4, 8], Prolog [11].

3.2. Arquitectura de un Sistema Basado en Reglas

En la Fig. IV-1 Se presenta la arquitectura de un sistema basado en reglas. Se distinguen 4 componentes principales, los cuáles interactúan entre si para llevar a cabo la

ejecución de un programa: el motor de inferencia (a su vez se subdivide en un analizador de patrones y una agenda), la memoria de trabajo, las reglas base y el motor de ejecución [8]. A continuación, se realiza una breve descripción de cada componente:

- **Motor de inferencia (Inference Engine)**: Es el encargado de aplicar las reglas a la información. Controla el proceso por el cual se aplican las reglas a la información almacenada en la memoria de trabajo para obtener los resultados del problema (salida del sistema).

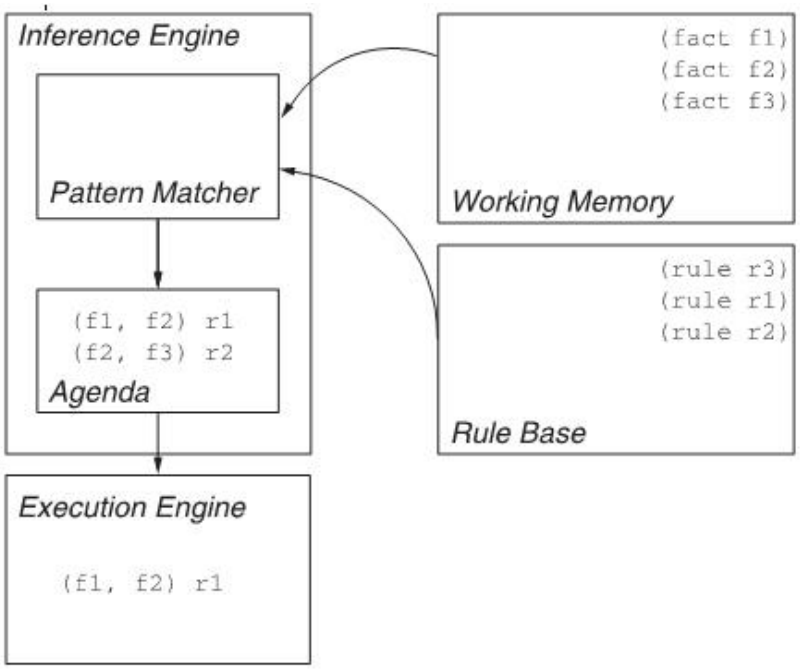

Fig. IV- 1. Arquitectura de un sistema basado en reglas.

- **Conjunto de reglas base (Rule Base)**: Contiene todas las reglas que el sistema conoce. Representa el conocimiento heurístico.

- **Memoria de trabajo (Working Memory)**: La memoria de trabajo almacena la información con la que el sistema basado en reglas trabaja. En esta es posible almacenar tanto las premisas como las conclusiones. Uno o más índices, similares a los utilizados en bases de datos, son mantenidos por el motor de

reglas para hacer de la búsqueda en la memoria de trabajo una operación rápida.

- **Analizador de patrones (Pattern Matcher)**: El motor de inferencias debe decidir qué reglas disparar y en qué momento. Para eso hace uso del *pattern matcher*. El propósito de este último es decidir qué reglas ejecutar bajo un eventual estado de la memoria de trabajo.

- **Agenda**: Una vez que el motor de inferencia decide qué reglas debe disparar, tiene que decidir cuáles de estas ejecutar primero. La lista de las posibles reglas a ejecutar es almacenada en la agenda. Esta es responsable de usar una estrategia de resolución de conflictos para decidir qué regla tiene la prioridad más alta y debe ser ejecutada primero.

- **Motor de ejecución (Execution Engine):** El motor de ejecución es el componente del motor de inferencia que ejecuta la regla. En los sistemas de producción clásicos las reglas pueden agregar, eliminar y modificar hechos en la memoria de trabajo. En los motores de reglas modernos, la ejecución de una regla puede generar un amplio rango de efectos. Algunos de estos ofrecen un lenguaje de programación específico para determinar qué ocurre cuando una regla es ejecutada.

3.3. Desarrollo un Sistema Basado en Reglas

El proceso de desarrollo de un sistema basado en reglas, incluye las siguientes etapas [8].

1. **Ingeniería del conocimiento (Knowledge engineering)**: Es la primera etapa del proceso, en la que se recolecta la información mediante la cuál se derivan las reglas. Las personas encargadas de realizar esta tarea se las conoce como ingenieros de conocimiento (*knowledge engineer*).

2. **Estructuración de datos (Structuring data)**: Una vez que toda la información ha sido recolectada, las tareas concernientes a la programación comienzan. Lo aconsejable es analizar la información y diseñar estructuras de datos convenientes, que ayuden a que la implementación de las reglas sea una labor clara y directa.

3. **Testing**: El testing en etapas tempranas de desarrollo ayuda a identificar errores que pueden resultar sumamente costosos si no son detectados a tiempo. Un sistema que es probado en cada etapa de su desarrollo será naturalmente más robusto y confiable que uno que no es probado hasta el final. Se recomienda, antes de escribir un conjunto de reglas, generar un script automático para probarlas. Es importante que los scripts de prueba sean automáticos, para que no requieran de mayor esfuerzo. Este ejercicio resulta en lo que se conoce como desarrollo dirigido por casos de prueba (*test-driven development*).

4. **Desarrollo de la interfaz (Interface building)**: Es importante tener en claro los límites del sistema. ¿Cómo interactuará el mismo con el mundo exterior, qué entradas necesitará para su procesamiento y cómo proveerá los resultados? Antes de comenzar con la codificación de las reglas, se aconseja realizar un diagrama de las conexiones entre el sistema y los componentes con los que se relaciona.

5. **Escribir las reglas (writing the rules)**: Una vez que las estructuras de datos están definidas, las interfaces especificadas, y los scripts de testeo existen, es momento de comenzar con la escritura de las reglas. Es aconsejable dividir las reglas en pequeños grupos. En general, un grupo de reglas es utilizado en un momento dado de la ejecución. Esta división permite generar un programa más sencillo de escribir y comprender.

6. **Desarrollo iterativo (Iterative development)**: Luego de desarrollar algunas reglas, es probable encontrarse en la situación que no se tiene la información necesaria para continuar con el desarrollo. Cuando esto ocurre, es necesario volver a la fuente de información y reiniciar el proceso de desarrollo, atravesando nuevamente sus etapas. El desarrollo de un sistema basado en reglas tiende a amoldarse a este proceso de naturaleza iterativa.

4. Jess (Java Expert System Shell)

Jess es un motor de reglas para el lenguaje Java, el cual fue desarrollado en los laboratorios Sandia al final de la década del 90 por Ernest J. Friedman-Hill [4, 8].

En la utilización de Jess, el programador especifica la lógica del programa en forma de reglas usando uno de dos formatos: el lenguaje de reglas de Jess (recomendado y utilizado en este trabajo) o XML. A partir de allí, se deberán proveer datos de entrada para que las reglas puedan trabajar. Cuando el motor se pone en funcionamiento, las reglas son ejecutadas. Estas pueden crear nuevos datos, o pueden realizar cualquier operación que el lenguaje Java puede hacer, ya que es posible hacer un llamado a cualquier función Java desde Jess.

El motor de reglas de Jess usa una versión optimizada del algoritmo Rete [5] para asociar las reglas con la memoria de trabajo. Este algoritmo sacrifica espacio para ganar velocidad, es así que Jess puede llegar a usar una cantidad considerable memoria. No existen comandos que permitan reducir la performance a favor de disminuir el uso de memoria. No obstante, un programa considerable en magnitud se ajustará fácilmente en la pila de 64 megabytes que Java usa por defecto [8, 14].

Aunque Jess puede correr como una aplicación independiente, frecuentemente se importa la biblioteca desde el código Java y se manipula usando una API. Es versátil en cuanto a su uso, se pueden construir aplicaciones usando solamente el lenguaje de reglas

de Jess, aplicaciones usando solamente las bibliotecas Jess desde Java, o una mezcla de ambas.

Es posible desarrollar código en lenguaje Jess en cualquier editor de texto, pero Jess provee un avanzado ambiente de desarrollo basado en la plataforma Eclipse [15]. La sintaxis de lenguaje Jess es simple y estándar. El lenguaje tiene pocos tipos de datos incorporados, entre ellos INTEGER, FLOAT, SYMBOL, STRING, y LONG. Además implementa estructuras de control simples, algunas de las cuales permiten transformar la información en código ejecutable. Todas las estructuras de control son, en realidad, funciones, Jess brinda casi 200 funciones predefinidas, y es posible definir nuevas usando el constructor "deffunction". También se permite modificar el comportamiento de las funciones predefinidas mediante el uso de "defadvice".

A continuación se listan los componentes básicos de este lenguaje y se provee una breve descripción de cada uno:

- ***deftemplate***: describe un tipo de hecho, a igual manera que una clase Java describe un tipo de objeto. Lista un conjunto de atributos, llamados *slots*, que el tipo de hecho puede contener. Se pueden realizar mapeos entre hechos Jess y clases Java mediante el uso conjunto de las sentencias *declare* y *from-class*. De esta manera, los *slots* del hecho estarán definidos por los atributos de la clase.

- ***defrule***: define una regla. Una regla está compuesta por una *LHS (Left-Hand Side)*, el símbolo "=>", y una *RHS (Right-Hand Side)*. En la parte izquierda - *LHS* – se encuentran uno o más elementos condicionales. La parte derecha - *RHS* - está compuesta por cero o más llamados a funciones. Los elementos condicionales son comprobados contra la memoria de trabajo de Jess. Si la comprobación es exitosa, el código en la parte derecha de la regla es ejecutado.

- ***deffunction***: define una función escrita en lenguaje Jess. Estas funciones pueden ser invocadas desde la línea de comandos, una regla, u otras funciones.

- ***defquery***: define una consulta, la cual consiste en una declaración opcional de una variable seguida por una lista de elementos condicionales. Las consultas (*queries*) son utilizadas para buscar hechos en la memoria de trabajo que satisfacen los elementos condicionales. Son de gran utilidad para obtener resultados una vez que las reglas que conforman la lógica del programa hayan sido ejecutadas.

Referencias

[1] Joel Jones. Abstract Syntax Tree Implementation Idioms. The 10th Conference on Pattern Languages of Programs, Sep. 8th-12th, 2003.

[2] Hadi A. S., Castillo E., Guti´errez J. M.(1997), Expert Systems and Probabilistic Network Models, Springer Verlag, New York.

[3] Iulian Neamtiu , Jeffrey S. Foster , Michael Hicks, Understanding source code evolution using abstract syntax tree matching, Proceedings of the 2005 international workshop on Mining software repositories, p.1-5, May 17-17, 2005, St. Louis, Missouri

[4] http://www.jessrules.com/

[5] Charles L. Forgy, "Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem," *Artificial Intelligence* 19 (1982): 17–37.

[6] Barr, A. and Feigenbaum, E. A. (1981), The Handbook of Artificial Intelligence, Volume I. William Kaufman, Los Altos, CA.

[7] http://www.rae.es/rae.html

[8] Friedman-Hill E. (2003). "Jess in Action". Manning Publications, ISBN 1-930110-89-8.

[9] Agnar Aamodt , Enric Plaza, Case-based reasoning: foundational issues, methodological variations, and system approaches, AI Communications, v.7 n.1, p.39-59, March 1994

[10] CLIPS (2008) http://www.ghg.net/clips/CLIPS.html

[11] Leon Sterling and Ehud Shapiro, *The Art of Prolog: Advanced Programming Techniques*, 1994, ISBN

[12] http://lazax.com/software/Mycin/mycin.html

[13] McDermott, J. R1: an expert in the computer systems domain. In Proceedings of the 1st Annual National Conference on Artificial Intelligence, pages 269-271. Stanford University, 1980.

[14] Microsoft's Rule Engine Scalability Results - A comparison with Jess and Drools

[15] Eclipse Homepage. http://www.eclipse.org. 2005.

[16] Chiang C., *An Introduction to Stochastic Processes and their Applications*, R.Krieger Publishing Company, 1968