Esta tesis doctoral contiene un índice que enlaza a cada uno de los capítulos de la misma.
Existen asimismo botones de retorno al índice al principio y final de cada uno de los capítulos.

Ir directamente al índice

Para una correcta visualización del texto es necesaria la versión de Adobe Acrobat Reader 7.0 o posteriores

Aquesta tesi doctoral conté un índex que enllaça a cadascun dels capítols. Existeixen així mateix botons de retorn a l'index al principi i final de cadascun deis capítols .

Anar directament a l'index

Per a una correcta visualització del text és necessària la versió d' Adobe Acrobat Reader 7.0 o posteriors.

Universidad de Alicante
Dpto. de *Ciencia de la Computación*
e Inteligencia Artificial

Tesis Doctoral

Sistemas de Razonamiento y Conocimiento Distribuido. Agentes Inteligentes

Faraón Llorens Largo

Dirigida por:
Dr. ***Ramón Rizo Aldeguer***

Noviembre de 2001

Als meus pares, perquè només
una sòlida base m'ha permés
arribar fins ací.

I com no, ... a Lirios

Tesis Doctoral

Sistemas de Razonamiento y Conocimiento Distribuido. Agentes Inteligentes

FARAÓN LLORENS LARGO

Departamento de *Ciencia de la Computación e Inteligencia Artificial*
Escuela Politécnica Superior
Universidad de Alicante

Noviembre de 2001

ÍNDICE GENERAL

Apéndices

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Agradecimientos

Esta es una página que quitaría del documento, de la que prescindiría y que rehusaba escribir. Pero como dicen que "es de bien nacidos ser agradecidos", aquí están mis agradecimientos. Seguro que líneas como las que voy a escribir inundan prólogos en cientos de libros y pueden sonar a tópicos, pero puestos a ello, os aseguro que las digo de corazón. No soy persona de palabra fácil, ni me gustan los discursos rimbombantes. No creo que los agradecimientos se tengan que plasmar con palabras, sino que se tienen que demostrar con hechos. Las personas que me han ayudado y han colaborado en que este trabajo que tienes ahora en tus manos haya podido ver la luz, saben que tienen mi reconocimiento.

Soy de los que dicen que los amigos los elijes pero los compañeros de trabajo te vienen impuestos. Pero que de todas formas hay que llevarse bien con todos. Sin embargo he de reconocer que he tenido la suerte de encontrar a grandes amigos en el trabajo. A todos vosotros, que empezasteis como compañeros de trabajo y os he acabado elijiendo como amigos, os quiero mostrar aquí mi agradecimiento.

Si se empieza con una ristra de nombres, se corre el riesgo de dejarse a alguien olvidado. Pido de antemano disculpas si eso ocurriese. En primer lugar quiero mostrar mi agradecimiento al departamento de Ciencia de la Computación e Inteligencia Artificial y a la Universidad de Alicante por poner a mi disposición los medios materiales y humanos que han posibilitado este trabajo de investigación. A todos mis compañeros del grupo de investigación VGIA porque, en cierta medida, son coparticipes de este trabajo. Especial mención, por razones personales, a Ramón,

Volver al índice/Tornar a l'índex

sin cuya dirección, apoyo, sabias palabras, pero sobre todo amistad, no existiría esta tesis. A Rosana por estar siempre ahí. A Mar, Chus y Paco Mora por su apoyo en las últimas fases del proceso. A Juanma y a Jero por seguir manteniendo los lazos de unión.

Por último, y no por orden de prioridad, agradecer a mis padres y hermanos por haber creado un entorno adecuado que ha formado mi inteligencia social y emocional. Esto me ha dado solidez para afrontar la vida con valentía. A Lirios por darme la estabilidad sentimental que ha facilitado que pueda dedicar el tiempo, a veces creo que excesivo, a mis expectativas profesionales. Muchas gracias, discúlpame por haberte descuidado y espero disponer ahora de tiempo para recompensarte por ello.

Con este trabajo culmina mi carrera formativa, que en algunos momentos considero que se ha dilatado excesivamente en el tiempo. Formación académica que empecé a los tres años cuando mis padres me llevaron por primera vez a la escuela, y que he finalizado treinta y siete años después con la lectura de esta tesis. Han sido casi cuatro décadas de acudir diariamente a las aulas. En un principio como estudiante y durante más de la mitad del tiempo compartiendo las labores de estudiante y profesor. No sé hacer otra cosa mas que aprender y enseñar lo que aprendo.

Este documento es el punto final de un trabajo. Pero en el momento que pasas una página, otra nueva aparece ante ti. Deseo que esto sea un punto y seguido, que sirva de sustento para proseguir con mi trayectoria profesional, cumpliendo con mi labor docente y desarrollando tareas de investigación, ahora sí, con pleno derecho.

Alcoy, 16 de noviembre de 2001

RESUMEN

La complejidad estructural de la gestión del conocimiento distribuido implica distintos aspectos, todos ellos relevantes y constituyentes de forma aislada en problemas con múltiples vertientes, para los cuales, dependiendo de la naturaleza del conocimiento, se tienen distintos enfoques de resolución. El concepto de *conocimiento distribuido* supone un salto cualitativo. Desde este punto de vista, en la tesis se aborda en especial la componente de integración y coordinación del conocimiento distribuido. Para ello es necesario afrontar aspectos de representación, razonamiento y coordinación. Centrándonos especialmente en la distribución, tanto física como lógica, en la tesis se propone un modelo a tal fin.

La **representación** del conocimiento que tenemos de un dominio o sobre un problema concreto a resolver es uno de los aspectos clave en cualquier aplicación que utilice inteligencia. Necesitamos recopilar dicho conocimiento e introducirlo en nuestro ordenador para resolver los problemas. Y el papel de los ingenieros del conocimiento es obtener ese conocimiento y hacerlo explícito. Pero no debemos perder de vista que una buena representación del conocimiento debe ser fácilmente entendible por los humanos y al mismo tiempo no ambigua y de fácil manipulación por parte del ordenador. Así podemos definir la representación del conocimiento como la aplicación de la lógica y la ontología a la tarea de construir modelos computables para algunos dominios. Se han desarrollado distintas formas de representar conocimiento, cada una de las cuales tiene sus ventajas y sus inconvenientes. Lo deseable sería disponer de un sistema de representación que incorporara distintas características. KIF ("Knowledge Interchange Format") es un lenguaje expresamente diseñado para el intercambio de conocimiento entre bases de conocimiento heterogéneas, basado en la lógica de predicados, que permite cierta flexibilidad en el lenguaje de representación al soportar la definición de objetos, funciones, relaciones, reglas y metaconocimiento. KIF es una representación interna del conocimiento, y en la medida en que distintos programas (agentes) puedan leerlo y escribirlo, dicho conocimiento será portable y reusable. Con ello podemos conseguir que el

conocimiento aparezca distribuido por el sistema y sea compartido por varios agentes.

El **razonamiento** es el proceso cognitivo por medio del cual utilizamos y aplicamos nuestro conocimiento, permitiéndonos pasar de una información a otra relacionada con esta. En el lenguaje cotidiano, lo más habitual es la formulación del conocimiento por medio de reglas. Además, a los sistemas basados en reglas es fácil incorporarles distintas características deseables en un sistema de razonamiento: tratamiento de incertidumbre, trabajo con información incompleta, posibilidad de rectificar y cambiar de opinión, manejo de creencias, etc. Por ello los sistemas expertos basados en reglas de producción han sido los más estudiados. En la actualidad el diseño de sistemas expertos suele llevarse a cabo con la ayuda de herramientas que ofrecen facilidades para modelar el conocimiento del dominio mediante reglas (CLIPS, Jess, Prolog, ...).

La complejidad del software aumenta de forma incesante y se demandan, cada vez más, programas que puedan interactuar entre ellos. Como resultado de estas necesidades surgen los agentes como área en rápido desarrollo, donde se aúnan un gran número de disciplinas diversas. El problema surge cuando dos seres inteligentes intentan comunicarse, ya que deben de utilizar el mismo lenguaje, estar de acuerdo en el significado de los símbolos de ese lenguaje, tener un mecanismo de comunicación para el intercambio de mensajes, no hablar al mismo tiempo, etc. En resumen, debe de existir **coordinación** entre ellos. De ahí la necesidad de un lenguaje que permita la intercomunicación entre agentes autónomos distribuidos (ACL "Agent Communication Language"). Un mensaje de ACL es un mensaje en KQML ("Knowledge Query and Manipulation Language") que consiste en una directiva de comunicación y un contenido semántico escrito en KIF. KQML es un lenguaje concebido al mismo tiempo como un formato de mensajes y como un protocolo que maneja esos mensajes para permitir a un programa identificar, conectarse e intercambiar información con otros programas. Intuitivamente, podemos decir que cada mensaje en KQML es una pieza de diálogo entre un emisor y un receptor, proporcionando el soporte para una amplia variedad de tipos de diálogos.

El documento está estructurado de la siguiente manera:

- En una primera parte se estudian y analizan los modelos actuales de representación del conocimiento (capítulo 1), los sistemas de razonamiento (capítulo 2) y con más detalle los sistemas basados en conocimiento (capítulo 3) y los sistemas de interacción entre agentes, sus arquitecturas y su utilización para distribuir y compartir conocimiento (capítulo 4).

- El capítulo 5 describe el modelo propuesto, basado en una arquitectura de agentes múltiples organizados en capas, incorporando las directivas necesarias para su viabilidad. Se aborda también su implementación. En el capítulo 6 se examinan aplicaciones de los agentes inteligentes en la red y se realizan ejemplos de la utilización del sistema implementado.

- Completamos con las conclusiones y las propuestas de futuro (capítulo 7).

- Cerramos con las citas bibliográficas a las que hacemos referencia en el documento.

Al final del mismo hay una serie de apéndices que pueden complementar la información aportada a lo largo del documento: introducción al concepto de inteligencia y breve descripción de la sintaxis de los lenguajes específicos utilizados.

RESUM

La complexitat estructural de la gestió del coneixement distribuït implica distints aspectes, tots ells rellevants i constituents de forma aïllada en problemes amb múltiples vessants, per als quals, depenent de la naturalesa del coneixement, es tenen distints enfocaments de resolució. El concepte de *coneixement distribuït* suposa un bot qualitatiu. Des d'este punt de vista, en la tesi s'aborda en especial la component d'integració i coordinació del coneixement distribuït. Per a això és necessari afrontar aspectes de representació, raonament i coordinació. Centrant-nos especialment en la distribució, tant física com a lògica, en la tesi es proposa un model a tal fi.

La **representació** del coneixement que tenim d'un domini o sobre un problema concret a resoldre és un dels aspectes clau en qualsevol aplicació que utilitze intel·ligència. Necessitem recopilar el dit coneixement i introduir-lo en el nostre ordinador per a resoldre els problemes. I el paper dels enginyers del coneixement és obtindre eixe coneixement i fer-lo explícit. Però no hem de perdre de vista que una bona representació del coneixement ha de ser fàcilment entesa pels humans i al mateix temps no ambigua i de fàcil manipulació per part de l'ordinador. Així podem definir la representació del coneixement com l'aplicació de la lògica i l'ontologia a la tasca de construir models computables per a alguns dominis. S'han desenrotllat distintes formes de representar coneixement, cada una de les quals té els seus avantatges i els seus inconvenients. El desitjable seria disposar d'un sistema de representació que incorporara distintes característiques. KIF ("Knowledge Interchange Format") és un llenguatge expressament dissenyat per a l'intercanvi de coneixement entre bases de coneixement heterogènies, basat en la lògica de predicats, que permet certa flexibilitat en el llenguatge de representació al suportar la definició d'objectes, funcions, relacions, regles i metaconeixement. KIF és una representació interna del coneixement, i en la mesura en què distints programes (agents) puguen llegir-lo i escriure'l, el dit coneixement serà portable i reusable. Amb això podem aconseguir que el coneixement aparega distribuït pel sistema i siga compartit per diversos agents.

El **raonament** és el procés cognitiu per mitjà del qual utilitzem i apliquem el nostre coneixement, permetent-nos passar d'una informació a una altra relacionada amb esta. En el llenguatge quotidià, el més habitual és la formulació del coneixement per mitjà de regles. A més, als sistemes basats en regles és fàcil incorporar-los distintes característiques desitjables en un sistema de raonament: tractament d'incertesa, treball amb informació incompleta, possibilitat de rectificar i canviar d'opinió, maneig de creences, etc. Per això els sistemes experts basats en regles de producció han sigut els més estudiats. En l'actualitat el disseny de sistemes experts sol dur-se a terme amb l'ajuda de ferramentes que oferixen facilitats per a modelar el coneixement del domini per mitjà de regles (CLIPS, Jess, Prolog, ...).

La complexitat del programari augmenta de forma incessant i es demanden, cada vegada més, programes que puguen interactuar entre ells. Com resultat d'estes necessitats sorgixen els agents com a àrea en ràpid desenrotllament, on s'unixen un gran nombre de disciplines diverses. El problema sorgix quan dos sers intel·ligents intenten comunicar-se, ja que deuen utilitzar el mateix llenguatge, estar de acord en el significat dels símbols d'eixe llenguatge, tindre un mecanisme de comunicació per a l'intercanvi de missatges, no parlar al mateix temps, etc. En resum, deu existir **coordinació** entre ells. D'ací la necessitat d'un llenguatge que permeta la intercomunicació entre agents autònoms distribuïts (ACL "Agent Communication Language"). Un missatge d'ACL és un missatge en KQML ("Knowledge Query and Manipulation Language") que consistix en una directiva de comunicació i un contingut semàntic escrit en KIF. KQML és un llenguatge concebut al mateix temps com un format de missatges i com un protocol que maneja eixos missatges per a permetre a un programa identificar, connectar-se i intercanviar informació amb altres programes. Intuïtivament, podem dir que cada missatge en KQML és una peça de diàleg entre un emissor i un receptor, proporcionant el suport per a una àmplia varietat de tipus de diàlegs.

El document està estructurat de la següent manera:

- En una primera part s'estudien i analitzen els models actuals de representació del coneixement (capítol 1), els sistemes de raonament (capítol 2) i amb més detall els sistemes basats en coneixement (capítol 3) i els sistemes de interacció entre agents, les seues arquitectures i la seua utilització per a distribuir i compartir coneixement (capítol 4).
- El capítol 5 descriu el model proposat, basat en una arquitectura de agents múltiples organitzats en capes, incorporant les directives necessàries per a la seua viabilitat. S'aborda també la seua

implementació. En el capítol 6 s'examinen aplicacions dels agents intel·ligents en la xàrcia i es realitzen exemples de la utilització del sistema implementat.

- Completem amb les conclusions i les propostes de futur (capítol 7).
- Tanquem amb les citacions bibliogràfiques a les que fem referència en el document.

Al final del mateix hi ha una sèrie d'apèndixs que poden complementar la informació aportada a el llarg del document: introducció al concepte d'intel·ligència i breu descripció de la sintaxi dels llenguatges específics utilitzats.

ABSTRACT

The structural complexity of the distributed knowledge management implies different aspects, all of them outstanding and constituent of isolated form in problems with manifold sides, for which, depending on the knowledge nature, different resolution approaches are obtained. The concept of *distributed knowledge* supposes a qualitative jump. From this point of view, in the thesis we undertake in special the integration and coordination component of the distributed knowledge. So, it is necessary to face up to aspects of representation, reasoning and coordination. In the thesis we propose a model concerned specially in the distribution, both physical and logical.

The **representation** of knowledge that we have of a domain or on a concrete problem to solve is one of the aspects key in any application that uses intelligence. We need to compile this knowledge and to introduce it in our computer to solve the problems. And the task of the knowledge engineers is to obtain that knowledge and to make it explicit. But we do not have to lose sight that a good knowledge representation must be easily understandable by the humans and at the same time nonambiguous and of easy manipulation for the computer. Thus we can define the knowledge representation like the application of the logic and the ontology to the task of constructing computable models for some domains. Different forms have been developed to represent knowledge, they have their advantages and their disadvantages. The desirable thing would be to have a representation system that incorporated different characteristics. KIF ("Knowledge Interchange Format") is a language specifically designed for knowledge interchange between heterogenous knowledge bases, based on the predicate logic, that allows to certain flexibility in the representation language, supporting the definition of objects, functions, relations, rules and metaknowledge. KIF is an internal representation of the knowledge, and as according that different programs (agents) can read and write it, this knowledge will be portable and reusable. With it we can obtain that the knowledge appears distributed by the system and be shared by several agents.

Reasoning is the cognitive process by means of we used and we applied our knowledge, allowing us to skip from one information to another related. In the natural language, most habitual it is the formulation of the knowledge by rules. In addition, to the ruled-based systems it is easy to incorporate different desirable characteristics for the reasoning system: uncertainty processing, working with incomplete information, possibility of rectifying and opinion changing, handling of beliefs, etc. For that reason the ruled-based expert systems have been extensively studied. At present, the design of expert systems is carried out with the aid of tools that offer facilities to model the domain knowledge by rules (CLIPS, Jess, Prolog...).

The complexity of software is increasing in a incessant form and, more and more, programs that can interact among them are demanded. As result of these necessities arises the agents like area in fast development, where they combine a great number of diverse disciplines. The problem arises when two intelligent beings try to communicate, as they must use the same language, they must agree in the meaning of the symbols of that language, they must have a mechanism of communication for the messages interchange, they must not to speak at the same time, and so on. In short, **coordination** among them must exist. Of there, the necessity of a language that allows the intercommunication between distributed independent agents (ACL "Agent Communication Language"). A ACL message is a message in KQML ("Knowledge Query and Manipulation Language") that consists in a communication performative and a semantic content written in KIF. KQML is a language conceived at the same time like a format of messages and as protocol that handles those messages to allow a program to identify, to connect and to interchange information with other programs. Intuitively, we can say that each message in KQML is a piece of dialogue between a transmitter and a sender, providing the support for an ample variety of dialogue types.

The document is structured of the following way:

- In the first part we study and analyze the current models of knowledge representation (chapter 1), the reasoning systems (chapter 2) and with more detail the knowledge-based systems (chapter 3) and the systems of interaction between agents, their architectures and their use to distribute and to share knowledge (chapter 4).

- The chapter 5 describes the proposed model, based on an architecture of multiple agents organized in layers, incorporating the necessary guidance for its viability. Its implementation is also undertake. In the chapter 6 applications of the intelligent agents in the

network are examined and examples of the use of the implemented system are made.

- Following with the conclusions and the future proposals (chapter 7).
- And finally we close with the bibliographical citations to which we make reference in the document.

At the end of the same there is a series of appendices that can complement the information contributed throughout the document: introduction to the concept of intelligence and brief description of the syntax of the used specific languages.

Volver al índice/Tornar a l'índex

Capítulo 1

Análisis y Caracterización de los Métodos de Representación del Conocimiento

> To solve really hard problems, we'll have to use several different representations. This is because each particular kind of data structure has its own virtues and deficiencies, and none by itself would seem adequate for all the different functions involved with what we call common sense.
>
> Marvin Minsky

La representación del conocimiento que tenemos de un dominio o sobre un problema concreto a resolver es uno de los aspectos clave en cualquier aplicación que utilice Inteligencia Artificial (IA). El conocimiento lo adquirimos y transmitimos por medio del lenguaje. Es evidente que la evolución del lenguaje ha desempeñado un importante papel en el desarrollo humano. Aunque también es verdad que los humanos representamos buena parte de nuestro conocimiento de manera no verbal el lenguaje es el principal vehículo de comunicación. Pero el lenguaje natural que utilizamos es ambiguo y engorroso, y por ello, para trabajar de manera rigurosa necesitamos un lenguaje artificial o una representación formal de dicho conocimiento ([**KSL web, 2001**]). Así, para poder trabajar con conocimiento, primero deberemos formalizarlo y representarlo de una determinada manera. En este capítulo esbozaremos algunas técnicas de representación del conocimiento, para en el siguiente tratar los sistema de razonamiento. Aunque aquí los hayamos separado, son dos aspectos que van íntimamente ligados.

1.1 Introducción

¿Qué es conocimiento? podríamos decir que es el hecho o la condición de conocer algo que hemos adquirido por experiencia o asociación. Los seres humanos vamos adquiriendo conocimiento al ver, oír, tocar, sentir y oler el mundo que nos rodea. Y ese conocimiento lo almacenamos en nuestro cerebro. Pero para almacenarlo en los ordenadores debemos representar ese conocimiento de alguna forma. En palabras de Kosko, "el conocimiento reduce el espacio efectivo de búsqueda. Lo difícil es expresar los conocimientos mediante símbolos o números de tal manera que el ordenador pueda utilizarlos" ([**Kosko, 2000**]). En la medida en que necesitamos recopilar conocimiento e introducirlo en nuestro ordenador para resolver problemas es por lo que hablamos de Representación del Conocimiento. Y el papel de los ingenieros del conocimiento es obtener ese conocimiento y hacerlo explícito. Pero no debemos olvidar que una buena representación del conocimiento debe ser fácilmente entendible por la gente y al mismo tiempo no ambigua y de fácil manipulación por parte del ordenador.

La representación del conocimiento es una materia multidisciplinar que aplica teorías y técnicas de otros campos:

1. La **lógica** proporciona la estructura formal y las reglas de inferencia. Aporta los criterios para determinar si las sentencias son redundantes o contradictorias.

2. La **ontología**[1] define las clases de cosas que existen en el domino de aplicación (apartado 1.8). Evita que los términos y símbolos estén mal definidos y sean confusos.

3. La **computación** sustenta las aplicaciones que diferencian la representación del conocimiento de la pura filosofía. Permite que los dos puntos anteriores puedan ser implementados sobre ordenadores.

Así, podemos definir la *Representación del Conocimiento* como la aplicación de la lógica y la ontología a la tarea de construir modelos computables para algunos dominios ([**Sowa, 2000**]). En [**Davis *et al.*, 1993**] los expertos en representación del conocimiento Davis, Schrobe y Szolovits hacen una crítica sobre la situación de este campo y concluyen cinco principios básicos sobre representación del conocimiento y su papel en la inteligencia artificial:

1. La representación del conocimiento es un *sustituto*. Los objetos físicos, los eventos y las relaciones no pueden ser almacenadas di-

[1] La **ontología** es el estudio de la existencia

rectamente en los ordenadores, por eso son representados por símbolos que sirven como sustituciones de las cosas reales del mundo exterior.

2. La representación del conocimiento es un *conjunto de compromisos ontológicos.* En una base de conocimiento, la ontología determina las categorías de cosas que existen o pueden existir en un dominio de aplicación. Esas categorías representan los compromisos ontológicos del ingeniero del conocimiento.

3. La representación del conocimiento es una *teoría fragmentada del razonamiento inteligente.* Para soportar razonamiento acerca de las cosas en un domino, una representación del conocimiento debe también describir comportamiento se interacciones.

4. La representación del conocimiento es un *medio para la computación eficiente.* Junto a la representación del conocimiento, un sistema inteligente debe codificar conocimiento de forma que éste pueda ser procesado eficientemente en el ordenador.

5. La representación del conocimiento es un *medio de expresión humano.* Un buen lenguaje de representación del conocimiento debe facilitar la comunicación entre el ingeniero del conocimiento (que entiende sobre IA) y el experto en el campo concreto (que entiende de la aplicación).

Podemos decir que estamos intentando representar información acerca de las cosas del mundo real para almacenarlas en el ordenador. Para ello la información pasa por distintos niveles de representación. Basándose en [**McCarthy and Hayes, 1969**], Brachman ([**Brachman, 1979**]) propone cinco *niveles de representación del conocimiento*:

1. *Implementacional.* Nivel de estructura de datos (átomos, punteros, listas y cualquier otro conceptos de programación).

2. *Lógico.* Lógica simbólica (proposiciones, predicados, términos, cuantificadores y operadores lógicos).

3. *Epistemológico.* Nivel para definir tipos de conceptos con subtipos, herencia y relaciones estructurales.

4. *Conceptual.* Nivel para las relaciones semánticas, roles lingüísticos, objetos y acciones.

5. *Lingüístico.* Nivel de conceptos arbitrarios, palabras y expresiones del lenguaje natural.

Dentro de la inteligencia artificial, las distintas técnicas que se han ido desarrollado, se pueden agrupar en dos grandes líneas:

Procesamiento Simbólico : se utilizan símbolos para representar el conocimiento y el estado del mundo, y simulan el proceso cognitivo mediante algoritmos que manipulan esos símbolos.

Modelos Biológicos : el conexionismo está inspirado en como actúa el cerebro, caracterizándose por la computación paralela y adaptativa. También hay propuestas de modelos evolutivos y otras basadas en los comportamientos, muchas de ellas derivadas de los modelos animales. También se plantean modelos donde la inteligencia emerge de la dinámica de las interacciones con el mundo ([**Brooks, 1991c**]).

En una primera aproximación, podemos hablar de dos tipos de representación del conocimiento:

Representación Declarativa : una representación declarativa del conocimiento es aquella en la que dicho conocimiento está especificado, pero sin embargo, no viene dada la manera en que debe ser usado tal conocimiento. Por tanto, para utilizar el conocimiento de una representación declarativa debe disponerse de un procedimiento que especifique qué debe hacerse con el conocimiento y de qué modo debe hacerse.

Representación Procedimental : por contra, una representación procedimental del conocimiento es aquella en la que la información del control necesaria para utilizar el conocimiento se encuentra embebida en el propio conocimiento. Están intrínsecamente unidos el conocimiento y su manipulación.

En inteligencia artificial es habitual el uso de técnicas de representación declarativas, ya que al hacer una representación explícita del conocimiento éste es más fácil de modificar, al mismo tiempo que el disponer separadamente los algoritmos de control (razonamiento) nos permitirá optimizar y reutilizar los procedimientos de inferencia.

Otras formas de representar conocimiento, ampliamente utilizadas en los sistemas de bases de datos son la *representación relacional*, dónde el conocimiento es representado por tuplas (registros) de información y

éstas agrupadas en tablas; y la *representación jerárquica*, que se centra en las relaciones y los atributos compartidos entre los objetos o clases de objetos.

A lo largo del tiempo se han ido desarrollando distintas técnicas de representación del conocimiento, entre las que podemos destacar :

La Lógica de Predicados y las Reglas de Producción. La lógica de predicados representada en una notación en Forma Clausal se puede convertir en un demostrador lógico automático, con un método de razonamiento al que se le asocia un proceso de extracción de respuestas. A este tipo pertenece el lenguaje de programación lógica Prolog. Los sistemas basados en reglas de producción serán tratados detenidamente en el apartado Sistemas Expertos Basados en Reglas (3.1.2).

Las Redes Semánticas y los Marcos. Tiene su base en estructuras de información representativas de la memoria asociativa humana cuyo método de razonamiento se basa en la exploración de la estructura de información en busca de subestructuras que satisfagan la pregunta.

Los Modelos Conexionistas. Estos modelos consideran que el sistema mental está constituido por unas redes de activación, cada una de las cuales comprende un conjunto amplio de unidades de procesamiento, que simulan a las neuronas, y que se encuentran unidas por conexiones con pesos diferenciados.

Las Restricciones. Están formuladas como representaciones cualitativas de ecuaciones e inecuaciones numéricas, con el fin de servir de base a modelos de razonamiento para simulación de procesos físicos o para representar condiciones de unificación en la programación lógica (CLP "Constraint Logic Programming").

En los siguientes apartados vamos a describir brevemente algunos de los modelos de representación básicos más habituales. Algunas representaciones serán más útiles que otras en determinados problemas, pero no por ello más correctas. Cada una de ellas tienen sus ventajas y sus inconvenientes. Para una visión del actual estado del campo de la representación del conocimiento se puede acudir a [**Sowa, 2000**] y [**Brewka, 1996**].

1.2 Sistemas Basados en la Lógica

La *lógica* fue uno de los primeros lenguajes utilizados en inteligencia artificial para representar conocimiento, y no sólo como lenguaje sino como fundamento matemático de la misma ([**Genesereth and Nilsson, 1987**]). La *Lógica de Predicados* pone a nuestra disposición un lenguaje que nos permitirá formalizar expresiones del conocimiento humano haciendo explícitos los objetos y las relaciones, así como sus restricciones. Y dicho lenguaje se convierte en uno de los mecanismos de representación del conocimiento, con la ventaja de que además nos proporciona un método, la deducción matemática, para obtener nuevo conocimiento a partir del antiguo ([**Castel and Llorens, 1999**], [**Barwise and Etchemndy, 2000**], [**Reeves and Clarke, 1990**] y [**Nerode and Shore, 1997**]).

La lógica, como sistema formal que es, nos proporciona un lenguaje (aspecto *sintáctico*) para escribir sentencias, y una manera de interpretar (aspecto *semántico*) el significado de dichas sentencias. En la definición de ese *Lenguaje Formal* empezaremos por determinar el *alfabeto* (conjunto de símbolos) que utilizaremos y las frases (*fórmulas bien formadas*) que podremos construir con combinaciones autorizadas de sus símbolos. Así, ante una sentencia del lenguaje natural buscaremos sus componentes :

- Qué se afirma, es decir, las propiedades y relaciones que aparecen en la sentencia y que representaremos en forma de *predicados.*
- De quienes se afirma, es decir, los objetos o individuos a los que hace referencia la sentencia y que representaremos por medio de *términos.*

Cuando los predicados se aplican a un solo término se trata de propiedades o características de dicho objeto; cuando hacen referencia a varios sujetos suelen representar relaciones. Los objetos deben pertenecer a un dominio genérico o *Universo del Discurso*, y pueden ser :

Constantes : representan objetos concretos del dominio.

Variables : permiten referenciar cualquier elemento del universo.

Funciones : denotan objetos referenciados en función de otros objetos.

Así, con un predicado y los términos implicados en dicho predicado podemos formar *fórmulas atómicas* o elementales. Combinando dichas

fórmulas atómicas mediante las *conectivas lógicas* (conjunción "$\wedge$", disyunción "$\vee$", negación "$\neg$" e implicación "$\rightarrow$") obtendremos *fórmulas moleculares*. Por último, si queremos expresar la cantidad de objetos que satisfacen alguna condición utilizaremos los cuantificadores :

Universal "$\forall$" : todos los elementos del universo del discurso.

Existencial "$\exists$" : por lo menos un individuo del universo.

Hasta aquí hemos visto cómo representar conocimiento en lógica, pero además nos proporciona técnicas para obtener nuevo conocimiento, es decir derivar conocimiento del que ya tenemos. Los sistemas clásicos de deducción ([**Socher-Ambrosius and Johann, 1997**], [**Llorens and Mira, 2000**]) no nos proporcionan un método que nos garantice una solución, lo que propició estudios e investigaciones en *demostración automática* ([**Siekmann and Wrightson, 1983**]). En términos de representación del conocimiento, el problema puede plantearse como:

A partir de una base de conocimiento
(representada en forma de premisas $P_1, P_2, ..., P_n$)
contestar si puede afirmarse que Q es verdad

Que escrito en forma de deducción en la lógica de primer orden quedaría

$$P_1, P_2, ..., P_n \Rightarrow Q$$

Desde un punto de vista semántico, la deducción anterior puede sustituirse por el estudio de la insatisfacibilidad de

$$P_1 \wedge P_2 \wedge ... \wedge P_n \wedge \neg Q$$

Para simplificar la formulación sintáctica y uniformizar la representación, transformaremos las fórmulas a *Forma Clausal*, consistente en un conjunto de cláusulas cuantificadas universalmente. Puesto que la enumeración de interpretaciones en dominios cualesquiera se convierte en un objetivo imposible, trabajaremos en un dominio abstracto *Universo de Herbrand*, y aplicaremos los resultados derivados de su teorema :

La condición necesaria y suficiente para que un conjunto de cláusulas sea insatisfacible es que exista un conjunto de instancias básicas que sea insatisfacible

---

Robinson aportó a este proceso la *Regla de Resolución*, que permite a partir de dos instancias básicas con un literal común L pero con signo opuesto ($F_1 \vee L$ y $F_2 \vee \neg L$) obtener una nueva cláusula por unión de las partes restantes ($F_1 \vee F_2$). Esta regla es completa, es decir, si aplicando resolución no se obtiene la contradicción es que el conjunto es satisfacible. El concepto de *Unificador Más General* nos permite seleccionar los valores de las variables libres de forma que las instancias básicas elegidas sean resolubles. Con ello el proceso se podría resumir como :

```
Negar el objetivo a cumplir ¬O
Unirlo con la base de conocimiento C = BC ∪ ¬O
Repetir para el conjunto C de cláusulas resolventes
        elegir dos cláusulas,
        calcular el resolvente y
        añadirlo a C
Hasta deducir la cláusula vacía o no poder aplicar resolución
```

Algoritmo 1.1: Refutación por Resolución

Este proceso aún resultaba computacionalmente muy costoso, de forma que se plantearon propuestas de mejora del proceso de emparejamiento, como el modelo de *resolución lineal*. De esta forma podemos generar un árbol de deducción que puede explorarse en amplitud o en profundidad. El recorrido en amplitud es muy costoso en cuanto a utilización de memoria, pero respeta la completitud; si recorremos el árbol en profundidad sólo debemos almacenar información de la rama que estamos procesando, pero se puede elegir una rama infinita y el proceso no terminar pese a encontrase soluciones en otras ramas.

Hasta este momento únicamente estamos hablando de si la conclusión Q es deducible a partir de las premisa o no. Green propuso la incorporación a la regla de resolución de un proceso de *extracción de respuestas*.

Para reducir el coste, se dio un paso más, de manera que la aplicación de estas técnicas de deducción automática y extracción de respuestas se restringen a las *Cláusulas de Horn*, que son aquellas que tiene como mucho un literal afirmado. Esto nos permite representar las fórmulas del Cálculo de Predicados de una manera más intuitiva que la forma clausal ya que tiene tanto una semántica declarativa como una semántica procedimental más comprensible para el usuario.

Todos estos avances teóricos han dado lugar a un nuevo paradigma de programación, la *Programación Lógica* (**[Kowalski, 1986]** y **[Lloyd, 1993]**), cuyo lenguaje más representativo es *Prolog* (**[Clocksin and Mellish, 1987]**, **[Giannesini *et al.*, 1989]** y

[**Bratko, 1990**]). Por razones operativas, Prolog procesa los objetivos en el orden en que están formulados y elige las cláusulas a resolver en el orden en que están escritas. Además, se le han añadido predicados extralógicos que incrementan su eficiencia, pero que le restan expresividad.

1.3 Representaciones Derivadas de la Psicología

1.3.1 Redes Semánticas

Conjuntamente a los modelos comentados anteriormente, se desarrolló una línea de representaciones basadas en analogías con la memoria humana, iniciada por Quillian en 1968 ([**Quillian, 1968**]). Dicho trabajo planteaba las *Redes Semánticas*, como un modelo de la memoria asociativa humana basada en redes, de forma que el significado de un concepto depende del modo en que se encuentre conectado con otros conceptos. Una red semántica consiste en *entidades* y *relaciones* entre estas entidades. La podemos representar como un grafo dirigido cuyos nodos representan conceptos (objetos) y los arcos las relaciones de diverso tipo existentes entre los conceptos que conectan. La resolución de problemas sobre estas redes consiste en aplicar procesos de exploración que busquen subredes que encajen con la pregunta. Una forma típica de inferencia incorporada en las redes semánticas es la *herencia*. La representación del conocimiento basada en redes semánticas tiene el inconveniente de la excesiva complejidad que puede alcanzar una red que trate universos conceptuales complejos.

La representación del conocimiento en forma de redes es muy utilizada en Inteligencia Artificial, de forma que variando el significado dado a los nodos y a los enlaces de una red se pueden obtener métodos de representación de conocimiento adecuados para tareas muy diversas (redes asociativas, redes taxonómicas, redes bayesianas, ...). Las *Redes Bayesianas* están siendo ampliamente estudiadas y utilizadas como sistema de razonamiento con incertidumbre ([**Castillo *et al.*, 1996**]). El éxito de las redes está apoyado en que se pueden visualizar y formalizar utilizando grafos, estructura abstracta muy estudiada y utilizada en Ciencia de la Computación.

1.3.2 Marcos

Uno de los principales problemas de las distintas técnicas de representación vistas hasta ahora es la falta de estructuración de la información. En

1975 Minsky ([**Minsky, 1974**]) introdujo el concepto de *Marco* ("frame"), que constituye una forma estructurada de representación de conceptos y que pretende simular el contexto de las interacciones entre el sujeto y un conjunto de situaciones estereotipadas. La idea del autor es que cuando uno se enfrenta a una nueva situación extrae de su memoria una especie de "marco" o esquema que recuerda de situaciones pasadas y que le permite, cambiando pequeños detalles si es preciso, adaptarse a esta nueva situación. Un marco es, por tanto, un conjunto de datos estructurados que representan una situación estereotipada. Para ello se definen una serie de variables *atributo* o *ranuras* ("slots") que caracterizan el concepto y que toman valores en dominios predefinidos. Estas ranuras se pueden rellenar con valores por defecto. Junto con cada marco guardamos distintos tipos de información:

- información acerca de cómo usar el marco
- información acerca de lo que podemos esperar que ocurra
- información acerca de qué hacer si nuestras expectativas no se confirman

La organización del conocimiento puede hacerse en colecciones o *jerarquías* de marcos llamados *sistemas de marcos*. Una colección representa todos los prototipos conocidos y cuando se presenta un caso se asigna a alguno de ellos, en función de criterios de equiparación definidos en cada caso mediante recorrido secuencial de la colección. La organización en jerarquías se aplica cuando la colección tiene un tamaño demasiado grande para hacer eficiente el recorrido. Al igual que en las redes semánticas la forma más común de inferencia es la herencia: heredar propiedades de las entidades más generales es el principal mecanismo para deducir. De forma muy parecida, Schank y Abelson ([**Schank and Abelson, 1977**]) proponen una versión similar que utiliza guiones ("scripts").

La organización por *categorías* ha sido un enfoque tradicionalmente utilizado en psicología. Para ello se definen un conjunto de propiedades que son individualmente necesarias y en su conjunto suficientes. Así, decimos que una propiedad es necesaria si está presente en todos y cada uno de los individuos; y un conjunto de propiedades es suficiente si cada individuo que presente estas propiedades pertenece a la categoría en cuestión. Cuando hablamos de una propiedad hacemos referencia a un predicado; cuando nos referimos a un atributo estamos denotando al término genérico de esa propiedad; y si hablamos del valor estamos referenciando a la especificación concreta de ese atributo. Así por ejemplo, podemos caracterizar el concepto "soltero" como aquel individuo que

presente las siguientes tres propiedades: ser varón, adulto y no estar casado. En este ejemplo, una propiedad es "ser varón", que hace referencia al atributo "sexo" y que puede tomar el valor "varón". Vemos claramente que cada una de estas propiedades, por sí sola, no es suficiente para determinar si un individuo es soltero (por ejemplo ser varón no es suficiente para ser soltero, ya que un niño no está soltero), aunque conjuntamente sí son suficientes para determinar si un individuo pertenece a la clase. Por otro lado, cada una de las tres propiedades es necesaria para ser clasificado como "soltero".

1.3.3 Programación Orientada a Objetos

El paradigma de la *programación orientada a objetos* está inspirada, en parte, en los conceptos vistos en el apartado anterior y ofrece, por tanto, muchas de las características allí comentadas. La programación orientada a objetos intenta echar una mano al programador para que pueda trasladar al computador los problemas del mundo real de una forma mucho más natural. Para ello se basa en que en los problemas del mundo real se trata con una serie de elementos, a los que denomina *objetos*. En el paradigma orientado a objetos la ejecución de un programa se define como la interacción entre una serie de objetos, de manera muy similar a como lo harían en el mundo real, por supuesto, salvando las distancias.

La programación orientada a objetos utiliza dos conceptos clave, el de clase y el de objeto:

Clase : Es la representación de un conjunto de objetos que presentan una estructura y un comportamiento comunes. Por ejemplo, la clase `silla`, la clase `coche`, la clase `persona`, etc. La posibilidad de definir subclases y superclases hace que aumente la reusabilidad y modificabilidad del código generado, características muy deseables en un lenguaje de programación.

Objeto : Es la instancia concreta de una clase, por ejemplo la `silla que se encuentra primera al entrar a la cafetería por una puerta concreta`, el `coche marca X y matrícula A-xhjf784-CV`, etc.

En el paradigma orientado a objetos vamos a encontrar cuatro características que no tenemos disponibles habitualmente en otros paradigmas de programación:

1. La *encapsulación*, es decir, la posibilidad de agrupar bajo una misma entidad a unos datos y a las operaciones que trabajan con ellos. Si

en una tupla sólo podíamos guardar datos, ahora junto a esos datos vamos a poder tener también aquellas operaciones que saben qué hacer con ellos. Esto se consigue mediante las *clases*.

2. La *herencia* de clases, la cual nos servirá para expresar la similitud entre clases. Por ejemplo, si hemos creado una clase que representa la idea general de un vehículo, podemos crear una nueva clase heredando de ella y que represente la idea de vehículo con motor.

3. El *paso de mensajes*, el cual nos va a permitir que los distintos objetos que constituyan nuestra aplicación puedan comunicarse; esto lo hacen enviándose mensajes unos a otros. Un *mensaje* es lo que un objeto le pide a otro que realice, y cómo este objeto lo hace, es lo que se denomina un *método*. Al conjunto de mensajes a los que puede responder un objeto se le llama *interface*.

4. El *enlace dinámico*, permite al programador retrasar hasta el mismo instante de la ejecución de su aplicación, la decisión de qué método responderá a un determinado mensaje.

En los últimos años este tipo de lenguajes ha ido incrementando su popularidad, ya que la manera de tratar los datos los hace especialmente buenos para la programación de sistemas de gran dimensión. En nuestra implementación vamos a utilizar Java, que es un lenguaje orientado a objetos.

1.4 Modelos Conexionistas

Se basan en la concepción del sistema nervioso, constituido por componentes funcionales (neuronas) altamente organizadas y que son capaces de recibir y transmitir señales electroquímicas de forma muy especializada. Los *Modelos Conexionistas* consideran que el sistema mental está constituido por unas redes de unidades de procesamiento que se encuentran unidas por conexiones con pesos diferenciados ([**Hilera and Martínez, 1995**]).

En 1943 McCulloch y Pitts ([**McCulloch and Pitts, 1948**]) aplicaron los principios de la lógica simbólica a la descripción de la actividad neuronal y construyeron *redes neuronales artificiales* que podían reconocer patrones. En 1958 Rosenblatt propone los *perceptrones*, de forma que las redes podían modificar sus conexiones por medio del entrenamiento. Durante los siguientes años, el interés por la redes neuronales decayó. En los años ochenta se volvieron a reanudar las investigaciones sobre las redes neuronales. En las redes de Hopfield cada elemento tiene la facultad

de excitar e inhibir al resto de los elementos, de forma que una configuración inicial con unos pesos determinados puede evolucionar hacia una configuración estable.

Los modelos conexionistas poseen dos propiedades que se consideran fundamentales en cualquier sistema de procesamiento análogo al humano: la *distribución* y el *paralelismo*. Estos modelos ofrecen la ventaja frente a los modelos simbólicos de que no es necesario establecer descripciones lógicas explícitas sobre la naturaleza del problema. El ajuste del peso de las conexiones se realiza de forma reiterada de acuerdo con la retroalimentación que reciben del ambiente exterior y de la propia estructura interna de la red. Así, el sistema puede aprender a producir determinadas respuestas así como a presentar una respuesta completa ante la presencia parcial del patrón y, por tanto, la red puede seguir funcionando aún en el caso de que algunas de sus unidades se encuentren deterioradas.

Resumiendo, para explicar el comportamiento humano, el enfoque simbólico asume que los estados mentales están formados por representaciones simbólicas y reglas, mientras que el enfoque conexionista asume que únicamente hay activación y fuerza de conexión entre sus componentes.

1.5 Sistemas Basados en Restricciones

Habitualmente podemos representar las propiedades satisfechas por las soluciones a un problema en forma de ecuaciones e inecuaciones, de forma que dado un conjunto de variables definidas sobre dominios discretos y finitos, y un conjunto de restricciones definidas sobre subconjuntos de dichas variables, la solución al problema consistirá en encontrar una asignación de valores a las variables de forma que se satisfagan simultáneamente todas las restricciones dadas. Este tipo de problemas se conocen como *Problemas de Satisfacción de Restricciones* (CSP - "Constraint Satisfaction Problem"). En el área de la programación lógica se ha creado la Programación Lógica con Restricciones (CLP - "Constraint Logic Programming") que generaliza la programación lógica clásica incorporando ecuaciones e inecuaciones que condicionan las variables lógicas en forma adicional a las condiciones de unificación.

Los problemas de satisfacción de restricciones los podemos plantear como la asignación de valores a las variables para que, simultáneamente, satisfagan todas las restricciones. Lo podemos formular como <V,D,R>, donde:

- V es un conjunto finito de variables $\{ x_1, x_2, ..., x_n \}$

- D es un conjunto de dominios finitos $\{ D_1, D_2, ..., D_n \}$, donde $x_i \in D_i$
- R es un conjunto finito de restricciones sobre subconjuntos de variables de V, $\{c_{ij}(x_i, x_j)\}$

Este modelo lo podemos definir de la siguiente forma :

Marco Conceptual : consideramos un conjunto de propiedades o atributos $x_1, x_2, ..., x_n$ que toman valores en un dominio finito $D_1, D_2,..., D_n$, respectivamente.

Conocimiento Declarativo (Restricciones) : conjunto de propiedades R_p que condicionan valores posibles de los atributos que están sujetos a dicha restricción.

Conocimiento Inferencial : proceso de búsqueda que parten de la hipótesis de asignar cualquier valor del dominio a cada variable y, mediante satisfacción de las distintas restricciones, obtener combinaciones de valores de los atributos que satisfacen la totalidad de condiciones R_p.

El problema de la satisfacción de restricciones es NP completo, por lo que la forma más generaliza de resolverlo es mediante iteración de procesos de resolución de inconsistencias locales, utilizando procesos de búsqueda con poda.

1.6 Representación de Incertidumbre

En el mundo real, desafortunadamente, lo habitual es que no tengas toda la información, no haya una absoluta seguridad sobre lo que sabes y que, sobre la marcha, tomes decisiones con la información que tienes y seas capaz de rectificar ante nuevos datos. Por ello se desarrollaron lo que podríamos denomina *Técnicas de Razonamiento Aproximado*. Sus orígenes aparecen ya en los primeros prototipos de sistemas expertos tales como MYCIN o PROSPECTOR, que usan *medidas de certeza*. Se plantearon inicialmente teniendo en cuenta bases teóricas probabilísticas, de forma que incorporaban aspectos lógicos con aspectos de la teoría de probabilidades. Estos sistemas fueron aceptados por los especialistas como sistemas con rendimiento práctico similar al de personas expertas. No existe una técnica que sea la mejor para tratar con la incertidumbre, así como tampoco hay una definición clara y un único tipo de incertidumbre. Veremos en primer lugar que entendemos por incertidumbre y distintas

situaciones donde aparece, para pasar a continuación a describir algunas técnicas desarrolladas en el campo de la Inteligencia Artificial que nos permiten trabajar con este tipo de información (incierta, imprecisa, cambiante, indeterminada, ambigua, con grados de creencia, ...). El conocimiento de distintas técnicas es interesante ya que nos permitirá usar en cada momento aquella técnica que mejor capture el tipo de incertidumbre al que nos enfrentemos. En los últimos tiempos han ido apareciendo propuestas de lógicas que manejan incertidumbre ([**Gabbay *et al.*, 1994**] y [**Westerstahl, 2000**])

Podemos considerar la *incertidumbre*, no como una propiedad de la información, sino como un estado mental de un determinado agente:

La incertidumbre no está en las cosas sino en nuestra cabeza: la incertidumbre es el desconocimiento

- incertidumbre acerca de la existencia de un objeto
- incertidumbre acerca del valor de una propiedad
- incertidumbre acerca de la verdad de una sentencia
- incertidumbre acerca de la acción a realizar

Así, cuando hablemos de una información incierta nos estaremos refiriendo a una información que inducirá al agente a un estado de incertidumbre. La información incierta puede ser de distinto tipo:

Imprecisa "El objetivo está en la habitación 1"

Vaga "El objetivo está en el centro de la habitación 1"

Cambiante "El objetivo estaba hace un momento en las coordenadas (8,11)"

Necesitamos, por tanto, formalismos que nos permitan representar y manejar esta información incierta al nivel de detalle que necesitemos. Se han ido desarrollando distintos modelos que permiten la manipulación de información con distinto tipo de incertidumbre:

- **Modelos No Monótonos**, que permiten rectificar ante nueva evidencia.
- **Modelos Temporales**, que tienen en cuenta la incertidumbre asociada con el tiempo, ya que la situación de un momento dado no tiene porque ser la misma un instante después.
- **Modelos Probabilísticos**, que plantean procesos de razonamiento acordes con el contexto de la Teoría de Probabilidades.

- **Modelos Posibilísticos**, que plantean formas de razonamiento acordes con el contexto de la Lógica a las que se le asocian unas medidas numéricas de posibilidad : Lógica Difusa.

Se ha debatido mucho sobre la relación entre la probabilidad y la teoría difusa. Unos autores mantienen sus diferencias; otros argumentan que no hay nada que se pueda hacer con las herramientas de la lógica difusa que no se pueda hacer con probabilidades. Hay que resaltar que aunque la función de pertenencia difusa (función de posibilidad) tiene algún parecido con la función de probabilidad hay diferencias esenciales entre ambos conceptos ([**Bezdek, 1993**]). Ambos conceptos son diferentes, de hecho, el concepto de conjunto difuso es por naturaleza no estadístico:

- la pertenencia difusa representa similitud entre objetos con propiedades definidas imprecisamente de manera que determinarán un *grado de verdad* que sólo cambiará si cambiamos el mundo;
- mientras que la probabilidad nos da información acerca de frecuencias relativas de forma que determinan un *grado de creencia* que cambiará al contar con nuevas evidencias.

1.7 Formato para el Intercambio de Conocimiento (KIF)

Hemos visto distintas formas de representar conocimiento, cada una de las cuales tiene sus ventajas y sus inconvenientes. Lo deseable sería disponer de un sistema de representación que incorporara las distintas características que hemos ido enunciando. Últimamente está emergiendo una representación del conocimiento que se está convirtiendo en un estándar, KIF ("Knowledge Interchange Format") ([**Genesereth and Fikes, 1992**],[**Genesereth and Fikes, 1998**] y [**KIF web, 2001**]), desarrollado por el grupo de trabajo "Knowledge Sharing Effort" ([**KSE web, 2001**]). KIF es un lenguaje expresamente diseñado para el intercambio de conocimiento entre bases de conocimiento heterogéneas, basado en la lógica de predicados (con tipado), que permite cierta flexibilidad en el lenguaje de representación al soportar la definición de objetos, funciones, relaciones, reglas y metaconocimiento. KIF es una representación interna del conocimiento, y en la medida en que distintos programas (agentes) puedan leerlo y escribirlo, dicho conocimiento será portable y reusable. Con ello podemos conseguir que el conocimiento aparezca distribuido por el sistema y sea compartido por los distintos agentes.

La sintaxis del lenguaje es similar al LISP, aunque no es un lenguaje de programación. Ha sido diseñado como una versión del cálculo de predicados con un conjunto de caracteres y una sintaxis que simplifican su introducción en un ordenador. Es un lenguaje para intercambiar información, de forma que un agente puede escribir su conocimiento en formato KIF y otro agente leerlo, aunque sus representaciones internas de conocimiento sean incompatibles.

Al tener una sintaxis similar al LISP las frases están formadas por listas equilibradas de paréntesis, con operadores y funciones en posición prefija. Las variables empiezan por el carácter "?". Veamos un ejemplo, utilizando primero la notación sin tipos y luego la notación tipada:

Sentencia:

Hay un gato sobre la alfombra

Cálculo de Predicados:

$\exists$ x $\exists$ y (gato(x) $\wedge$ alfombra(y) $\wedge$ estasobre(x,y))

KIF (sin tipos):

```
(exists (?x ?y) (and (gato ?x) (alfombra ?y) (estasobre ?x ?y)))
```

KIF (con tipos):

```
(exists ((?x gato) (?y alfombra)) (estasobre ?x ?y))
```

Una característica destacable de KIF es la posibilidad de codificar conocimiento acerca del conocimiento (*metaconocimiento*) mediante los operadores "'" y ",".

Jorge está interesado en recibir información sobre salarios

```
(interested jorge '(salario ,?x ,?y ,?z))
```

1.8 Ontología

El objetivo de la *ontología* es el estudio de todas las clases de entidades que conforman nuestro mundo, de la existencia en sí. Así, llamaremos una ontología a un catálogo de los tipos de cosas que asumimos que existen en un domino de interés, desde la perspectiva de una persona que usa un lenguaje con el propósito de hablar acerca del dominio. Aunque el término procede de la Filosofía [2], en los últimos tiempos ha sido reutilizado en el campo de la Inteligencia Artificial. Así podemos definir la ontología como ([**Neches *et al.*, 1991**]):

[2]**Ontología**: parte de la filosofía que estudia el ente en cuanto tal.

Una ontología define los términos y relaciones básicas que forman parte del vocabulario de una determinada área, así como las reglas para combinar términos y relaciones para definir extensiones del vocabulario

---

Los elementos de una ontología son ([**Gómez-Pérez and Benjamins, 1999**] y [**Campoy, 2001**]):

Conceptos : cualquier cosa sobre la que podamos emitir un juicio o comentario.

Relaciones : representan diferentes tipos de interacción entre conceptos del dominio.

Funciones : son un caso especial de relaciones.

Instancias : representan cada uno de los elementos.

Axiomas : sentencias que son siempre verdaderas en ese dominio.

Las dos principales fuentes de categorías ontológicas son:

Observación : proporciona conocimiento del mundo físico.

Razonamiento : da sentido a las observaciones generando un marco de abstracción llamado *metafísica.*

La elección de las categorías ontológicas es el primer paso en el diseño de una base de conocimiento. Esta selección de categorías determinará todas las cosas que pueden ser representadas en la aplicación informática.

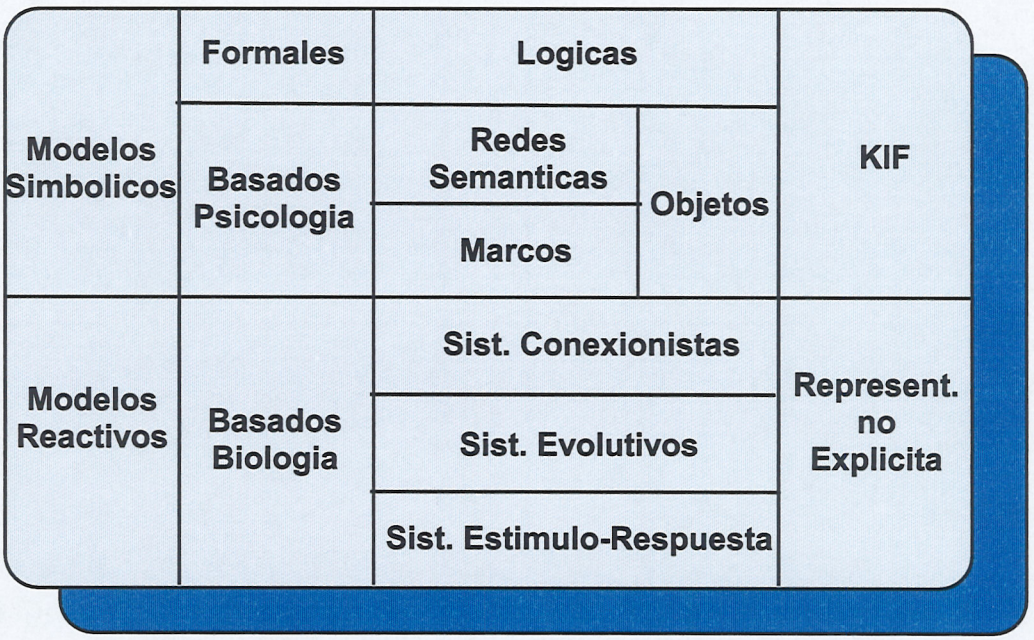


Figura 1.1: Esquema de Modelos de Representación del Conocimiento

Capítulo 2

Análisis y Caracterización de los Sistemas de Razonamiento

> Hay ensayos más radicales, que desisten por completo de preocuparse por la completitud o la coherencia y tratan de reproducir el razonamiento humano, con todas sus incoherencias. Estas búsquedas ya no tienen por meta aportar un sólido apuntalamiento a la matemática, sino exclusivamente estudiar los procesos del pensamiento humano.
>
> *Gödel, Escher, Bach. Un Eterno y Grácil Bucle.*
> Douglas R. Hofstadter

En este capítulo vamos a hablar de distintos sistemas de razonamientos, para en el próximo centrarnos en los sistemas basados en reglas. Cada método de representación tiene su propio mecanismo de inferencia asociado. Así, la lógica dispone de la deducción, la programación lógica utiliza la regla de resolución, la representación por reglas utiliza el encadenamiento hacia atrás o hacia adelante, los marcos utilizan la herencia, etc. Al mismo tiempo, cada método ha utilizado y desarrollado lenguajes de programación (paradigmas de programación) adecuados a su forma de resolver los problemas. Con ello queremos hacer notar que no podemos separar representación de razonamiento, ya que la elección tomada en un aspecto influirá en el otro.

Vamos en primer lugar a hacer una breve introducción al concepto de razonamiento, para pasar posteriormente a describir algunas técnicas básicas de razonamiento. Más información sobre modelos de razonamiento se puede encontrar en [**Fagin *et al.*, 1996**], [**Cuena, 1996**] [**González, 1998**] y [**Hofstadter, 1998**], entre otros.

2.1 Introducción

El *razonamiento* es el proceso cognitivo por medio del cual utilizamos y aplicamos nuestro conocimiento, permitiéndonos pasar de una información a otra relacionada con esta. Sin la posibilidad de hacer inferencias, el sistema de procesamiento de información se vería obligado a tener que definir todas las situaciones puntuales y específicas con las que se debe enfrentar.

Las sentencias de las cuales partimos en el proceso de razonamiento se llaman *premisas* y a la sentencia a la cual llegamos se le denomina *conclusión*. Las premisas junto a la conclusión forman el *argumento*. El concepto lógico de deducción correcta dice que de premisas verdaderas debemos obtener conclusión verdadera, es decir, no podemos aceptar que las premisas sean verdaderas y la conclusión falsa.

Clásicamente se distingue entre dos tipos de razonamiento: razonamiento deductivo y razonamiento inductivo. En el Razonamiento Deductivo se parte de unas premisas para alcanzar una conclusión que necesariamente se debe seguir de ellas, mientras que en el Razonamiento Inductivo alcanzaremos una conclusión que vendrá más o menos apoyada por las premisas. Veámoslo con más detenimiento:

- En un **razonamiento deductivo** la conclusión se sigue necesariamente de las premisas, o lo que es lo mismo, un razonamiento deductivo es correcto si es imposible que las premisas sean verdaderas y la conclusión falsa. Es por ello que decimos que el razonamiento deductivo es un proceso que va de lo general a lo particular o "hacia abajo". Las conclusiones deductivas son en cierto modo tautológicas (cierto o falso, todo o nada) y únicamente reflejan información contenida en las premisas. Las matemáticas son deductivas, en el sentido de que los hechos matemáticos surgen de las hipótesis.

- En un **razonamiento inductivo** las premisas apoyan o sugieren la conclusión. Por ello decimos que es un proceso que se llega a lo general a partir de lo particular o "hacia arriba". Diremos que un razonamiento inductivo es fuerte si es improbable que la conclusión sea falsa cuando las premisas sean verdaderas. Hablamos por tanto de fuerza del argumento y esto es cuestión de grados. Por tanto las conclusiones inductivas son probabilísticas y van más allá de la propia evidencia expresada en las premisas. Las ciencias experimentales son inductivas ya que la hipótesis es consecuencia de los hechos, es decir, la hipótesis surge después de observar algunos hechos o reflexionar sobre ellos.

2.2 Razonamiento Condicional

La formulación del conocimiento por medio de reglas (*si ... entonces*) es la más habitual en el lenguaje cotidiano. Y la lógica nos facilita dos reglas muy utilizadas y conocidas para realizar inferencias sobre este tipo de conocimiento:

- *Modus Ponens*: si tenemos que $A \rightarrow B$ y sabemos que A, entonces podemos deducir que B.
- *Modus Tollens*: si tenemos que $A \rightarrow B$ y sabemos que $\neg B$, entonces podemos deducir que $\neg A$.

Por ejemplo, supongamos que conozco que mi madre "siempre que llueve se queda en casa". Si miro por la ventana y veo que está lloviendo, puedo deducir que mi madre estará en casa (modus ponens). También, si llamo por teléfono y no contesta (no está en casa) puedo inferir que no estará lloviendo (modus tollens). En cambio seria erróneo asumir que llueve sólo porque mi madre esté en casa (falacia de afirmación del consecuente) así como deducir que no está en casa si veo por la ventana que no está lloviendo (falacia de negación del antecedente). En la figura 3.1 (página 49) vemos gráficamente la aplicación de estas dos reglas al proceso de inferencia.

Los sistemas basados en reglas de producción, creados por el lógico Post en 1943 como mecanismo de computación, encontraron una amplia aplicación en distintos campos: inteligencia artificial, teoría de autómatas, lenguajes de programación, gramáticas formales, etc. Veamos ahora a grandes rasgos los sistemas basados en reglas de producción. Podemos definir las *Reglas de Producción* como reglas cuya estructura es:

`situación` $\rightarrow$ `acción`

Y que se pueden expresar como :

Si una situación intermedia en el proceso de resolución de un problema verifica las precondiciones p_1, p_2, ..., p_n,

Entonces deben realizarse una serie de acciones a_1, a_2, ..., a_m

$$p_1, p_2, \ldots, p_n \rightarrow a_1, a_2, \ldots, a_m$$

El conocimiento que tenemos lo expresaremos con un conjunto de reglas, que llamaremos base de conocimiento. El modelo de razonamiento

para resolver problemas es muy similar al usado en la computación clásica mediante procesos algorítmicos, ya que esta base de conocimiento puede entenderse como formado por un conjunto de procedimientos (uno por regla), quedando libre la forma con la que deben encadenarse para llegar a producir una solución al problema. Este proceso de razonamiento puede dividirse en 2 etapas :

Reconocimiento en donde, en función del estado actual, se identifican las reglas de producción que son aplicables, es decir, aquellas en las que se cumplen las precondiciones.

Actuación en la que se aplican algunas o todas (en paralelo) las reglas que satisfacen las precondiciones.

El proceso se repite hasta que se llega a una situación en la que se satisfacen las condiciones de la solución. Este proceso de razonamiento se denomina *guiado por los datos* o *razonamiento hacia adelante*; se trata de un razonamiento deductivo. El número de reglas a disparar depende de la estrategia de control. Por otro lado, hablamos de razonamiento *guiado por objetivos* o *razonamiento hacia atrás* cuando el método de inferencia opera desde las conclusiones hacia los objetivos; se trataría de un razonamiento inductivo. Uno y otro tienen sus virtudes y sus inconvenientes, y serán analizados con más detenimiento en el apartado 3.2.2.

2.3 Razonamiento Probabilístico

En muchas ocasiones las premisas no nos permiten extraer conclusiones definitivas, bien porque no se dispone de toda la información necesaria o bien porque dicha información únicamente nos permite predecir posibles resultados. Las conclusiones alcanzadas serán, por tanto, *probables*. En nuestra vida diaria continuamente estamos tomando decisiones basadas en juicios de probabilidad. Por otro lado, cuando analizamos conocimiento acerca de nuestro entorno, las *cantidades relativas* (proporciones) son mucho más usuales que las *cantidades absolutas* (números). También se ha comentado antes que el razonamiento inductivo es básicamente probabilístico. Por todo ello, es interesante estudiar formas de razonamiento que incorporen estas características.

Así, por ejemplo, en el conocimiento humano es difícil encontrar afirmaciones universales. Cuando decimos que "Todos los pájaros vuelan" o que "Los pájaros vuelan" estamos afirmando que "la mayoría de los pájaros vuelan". Es decir, estamos haciendo una aserción sobre la cantidad

relativa de individuos del conjunto que vuelan entre el conjunto de individuos que son pájaros; no hablamos de ningún pájaro en particular. Las probabilidades estadísticas nos permiten modelar probabilidades definidas sobre conjuntos de individuos. Tales probabilidades no hacen referencia a las propiedades de individuos particulares del conjunto, sino que se refieren a propiedades del conjunto completo (como un todo).

Una forma de tratar con incertidumbre es por medio de un valor que represente el *grado de creencia* que el agente tiene en esa determinada sentencia. Pero este término es utilizado de distintas maneras por distintos autores. En la mayoría de los casos es un concepto subjetivo, en el sentido de que refleja el estado psicológico de un sujeto individual. Este concepto de grado de creencia puede ser tratado mediante las leyes del cálculo de probabilidades, mediante la asignación de un grado numérico de creencia entre 0 y 1.

2.3.1 Conceptos Básicos de Probabilidades

El marco formal en que se basa el razonamiento probabilístico es la *Teoría de Probabilidades*. Los orígenes de esta teoría se encuentran en los estudios de las leyes que rigen los juegos de azar de Fermat y Pascal en el siglo XVII y las obras de Bernoulli sobre el cálculo de probabilidades y de Laplace, a principios del siglo XIX, sobre una teoría analítica de las probabilidades.

En primer lugar deberemos definir el concepto de probabilidad. A pesar de que la teoría de la probabilidad está asentada y ampliamente aceptada, no existe una única interpretación del término probabilidad. Vamos a dar tres interpretaciones de la probabilidad que nos pueden ser útiles según el problema que tratemos:

- Interpretación como *frecuencia*: dado un suceso aleatorio la probabilidad de que ocurra ese suceso será la frecuencia relativa del mismo después de haberlo repetido un número razonablemente grande de veces en condiciones similares.

- Interpretación clásica (*objetiva*): dado un suceso aleatorio la probabilidad de que ocurra ese suceso será el cociente entre el número de casos favorables del suceso y el número total de resultados posibles, asumiendo que todos los resultados son igualmente verosímiles.

- Interpretación *subjetiva*: dado un suceso aleatorio la probabilidad que una determinada persona le asigna representa su propio juicio sobre la verosimilitud de que ocurra dicho suceso.

Como podemos ver las interpretaciones son distintas, pero la teoría de la probabilidad se ha desarrollado con independencia de la interpretación de probabilidad que se utilice en un problema particular. Las teorías y las técnicas serán válidas y nos servirán como herramienta para nuestro trabajo posterior.

El concepto de probabilidad nos permite indicar la *probabilidad a priori* o *incondicional* de que un determinado suceso ocurra. Se trata de un valor que le asignamos al suceso de no existir más información. En cuanto dispongamos de nueva información, nuevas evidencias, deberemos hablar de un nuevo valor esperado para dicho suceso. La probabilidad de un suceso S_1 cambiará cuando se sepa que otro suceso S_2 ha ocurrido. La denominaremos *probabilidad condicional* o a *posteriori* . El *teorema de Bayes* nos permite hallar la probabilidad de una hipótesis H a la vista de una nueva evidencia E. Esta probabilidad condicional vendrá dada a partir de la probabilidad a priori del suceso H y la medida en que se asocian ambos (hipótesis y evidencia) dada por $Pr(E|H)$ y $Pr(E|\neg H)$. Este teorema nos permite trabajar con probabilidades subjetivas al tratar con probabilidades a priori y probabilidades condicionadas, de forma que podemos incorporar las creencias o la evidencia existente a juicios tomados en situaciones de incertidumbre.

2.3.2 Psicología del Razonamiento Probabilístico

La psicología cognitiva ha estudiado en los últimos años el razonamiento probabilístico dada la enorme importancia que éste tiene en nuestro quehacer cotidiano y en gran cantidad de profesiones. Para ello se planteó estudiar cómo realizan las personas juicios bajo incertidumbre y cómo toman decisiones en lugar de qué deciden. En la vida diaria es difícil poder asignar probabilidades objetivas a las situaciones, bien porque no podemos repetir el suceso el número de veces necesario o las distintas alternativas posibles no son equiprobables. Por ello, en muchas ocasiones se utilizan probabilidades subjetivas procedentes de la información de que se dispone en ese momento o de las creencias y opiniones de sujeto que formaliza el problema.

Los experimentos realizados en psicología para comparar la conducta ideal (formalizada por el teorema de Bayes) y la actuación de las personas al emitir juicios de probabilidad apuntaron una tendencia conservadora en la emisión de dichos juicios de forma que la integración de la nueva evidencia para hallar la probabilidad a posteriori estaba más cercana a la probabilidad a priori que la obtenida aplicando el teorema de Bayes. Los psicólogos matemáticos Tversky y Kahnemann sostienen que los humanos normalmente no analizamos los acontecimientos coti-

dianos mediante listas exhaustivas de probabilidades ni elaboramos los pronósticos finales como combinación de parámetros estadísticos. Por el contrario apuntan que utilizamos unas "reglas de andar por casa" para emitir nuestros juicios ([**Kahneman *et al.*, 1982**]). A estas reglas las denominan *heurísticos*, y permiten simplificar la tarea de asignar probabilidades y de predecir. Su uso podría venir justificado por limitaciones cognitivas como la capacidad de la memoria a corto plazo ya que no podemos tener en cuenta toda la información al emitir un juicio. Estos autores proponen tres heurísticos principales:

Heurístico de representatividad : asignar a la hipótesis presentada una probabilidad en función de su parecido con el prototipo de la categoría.

Heurístico de accesibilidad : asignar a la hipótesis una probabilidad en función de la facilidad con que puede recordarse. Se basa en que lo más frecuente es también lo más disponible en la memoria y, por tanto, su recuperación es más fácil y rápida.

Heurístico de anclaje y ajuste : asignamos a la hipótesis una probabilidad en función de un valor inicial (anclaje) que posteriormente vamos ajustando.

Aunque estos heurísticos son válidos en muchos casos, es fácil comprobar que también nos pueden llevar a errores. Por ejemplo el heurístico de representatividad ignora información irrelevante para la semejanza pero fundamental desde el punto de vista estadístico (tamaño de la muestra, probabilidad a priori, ...). El heurístico de accesibilidad no tiene en cuenta que con mucha frecuencia la información que mejor puede recordarse es la más reciente, la más impactante o la más familiar. Y por último, el heurístico de anclaje y ajuste está altamente influenciado por el valor sobre el que se produce el anclaje y por tanto la respuesta final estará sesgada hacia este valor, y dicho valor puede estar sugerido, por ejemplo, por la formulación particular del problema. Otra crítica que se le hace a los heurísticos es que no existe un criterio para determinar cuando se aplica uno u otro. Es posible que los heurísticos se apliquen sucesivamente y de forma combinada. Así, el anclaje se podría realizar tomando lo más representativo o lo más accesible en memoria e ir ajustando el juicio en dirección a la información posterior. De cualquier manera, estos heurísticos están estrechamente relacionados entre sí y algunas veces es difícil saber cuál es el que se está aplicando. También se les critica por la escasa validez externa de los estudios sobre heurísticos, ya que los resultados obtenidos en laboratorio no pueden extrapolarse a la realidad en la que se recibe información de forma muy variada y las consecuencias del juicio emitido son muy importantes. De todas formas, las estrategias

heurísticas parecen más adecuadas para explicar el razonamiento probabilístico humano que la teoría de probabilidades (teorema de Bayes) ya que en muchos casos conducen a resultados correctos o, al menos, aceptables y a un coste cognitivo menor.

Otro aspecto que se ha estudiado desde el campo de la psicología en relación al razonamiento bajo incertidumbre es la confianza del sujeto en el juicio emitido, de forma que el sujeto evalúa su ignorancia o certeza con respecto a la respuesta emitida. En general se observa un exceso de confianza con respecto a la proporción de aciertos. Sobreconfianza que aumenta a medida que aumentamos la dificultad de la pregunta. En cambio, muestran subconfianza cuando se enfrentan a preguntas más fáciles.

2.3.3 Lógicas Probabilísticas

Veamos cómo extender, con carácter general, los conceptos de la lógica para que manejen probabilidades. Podemos asignar a cada sentencia un valor numérico correspondiente al grado de creencia o probabilidad de dicha sentencia. Así, la asignación de una probabilidad 0 a una determinada sentencia indicaría nuestra creencia de que dicha sentencia es falsa, una probabilidad de 1 correspondería a una creencia de que la sentencia es verdadera y cualquier valor de probabilidad entre 0 y 1 harían referencia a grados intermedios de creencia en la verdad (o falsedad) de la sentencia. Pero en cualquier caso, la sentencia de hecho será verdadera o falsa, ya que estamos hablando de grados de creencia (a diferencia de la lógica difusa que maneja grados de verdad).

Así por ejemplo, podemos tener un grado de creencia de 0'8 de que mañana sea un día soleado, ya que las condiciones meteorológicas así nos lo hacen presagiar. Pero al levantarnos al día siguiente será verdadero o falso que haya salido el sol. No habrá valores de verdad intermedios. En cambio, sí que podemos tener grados de verdad (de cumplimiento) y por ejemplo el grado de verdad de la sentencia "hoy es un día soleado" es de 0'8 ya que aunque luce el sol hay unas pocas nubes.

Existen distintas propuestas de lógicas que manejen probabilidades: la lógica probabilística de Nilsson ([**Nilsson, 1986**]), el modelo de Dempster ([**Dempster, 1967**]) y Shafer ([**Shafer, 1976**]) y las propuestas formales de Bacchus ([**Bacchus, 1990**]) y Halpern ([**Halpern, 1990**]) son las más conocidas.

2.4 Razonamiento Posibilístico

Zadeh planteó en 1965 ([**Zadeh, 1965**]) el concepto de conjunto difuso para caracterizar la extensión de predicados vagos, donde es difícil identificar el borde que delimita la verdad de la falsedad de dicho predicado. Sin embargo, estas clases definidas imprecisamente juegan un papel importante en el pensamiento humano ([**Kosko, 1995**]). La Lógica Difusa permite diluir el blanco y el negro de la lógica clásica en los grises con que el sentido común percibe el mundo incierto en el que vivimos. Zadeh introdujo el término "fuzzy", que traducimos como difuso o borroso, aunque en los ámbitos académico y tecnológico está ampliamente aceptada la palabra *fuzzy*. La *Lógica Difusa* ([**Dubois and Prade, 1986**], [**McNeill and Freiberger, 1993**] y [**Klir and Yuan, 1995**]) se encargará del procesamiento de información imprecisa o ambigua y de modelar la incertidumbre.

Es un método de formalizar operaciones del razonamiento impreciso sobre conceptos imprecisos, típicos del razonamiento humano e incapaces de ser manipulados por la lógica convencional. En la gestión de la imprecisión, la lógica difusa se encarga de :

1. Representar de una manera adecuada el significado impreciso de aquellas proposiciones del lenguaje natural que lo tengan.
2. Controlar la transferencia de imprecisión de premisas a conclusiones cuando realicemos algún procedimiento de inferencia.

Sería demasiado ingenuo creer que la lógica difusa nos va a permitir describir el complejo mundo en el que vivimos, pero sí que representa una mejora respecto a la lógica bivaluada permitiéndonos manejar información imprecisa. La Lógica Difusa busca formular algunas reglas de inferencia aproximada, y para este propósito intenta formalizar el tratamiento lingüístico coloquial aplicado a conceptos imprecisos tales como "muy", "más o menos", "demasiado", etc. Fue diseñada para permitir trabajar no sólo con métodos cuantitativos sino también con cualitativos, aspecto diferencial con el razonamiento humano. Podemos distinguir dos tipos de conocimiento : *conocimiento objetivo*, que ha sido utilizado habitualmente en la formulación y resolución de problemas con modelos matemáticos ; y *conocimiento subjetivo*, representado por información lingüística y normalmente imposible de cuantificar con herramientas matemáticas tradicionales. Ambos tipos de conocimiento pueden ser conjugados para resolver problemas reales por medio de la lógica difusa.

La lógica difusa tiene características que la hacen adecuada para tratar cierto tipo de incertidumbre. El conocimiento que tenemos podemos fácilmente implementarlo en reglas de producción difusas donde podemos

incorporar heurísticas y manejar incertidumbre. Por otro lado la utilización de variables lingüísticas (información cualitativa) nos permitirá un mejor diseño del sistema no lineal ([**Mendel, 1995**]).

La Lógica Difusa esta formalizada sobre la base de una lógica multivaluada con las conectivas lógica típicas y valores en el intervalo [0,1] indicando el *grado de verdad* (y *grado de falsedad*) del conocimiento representado por la fórmula. Así, establecemos una gradación continua del valor de verdad desde 0 (totalmente falso) a 1 (totalmente cierto).

La inferencia basada en la lógica clásica sólo es posible cuando los datos coinciden exactamente con la premisa dada, pues en caso contrario no podemos efectuar razonamiento alguno. En cambio, la inferencia difusa es posible incluso cuando el significado del hecho difiere ligeramente del conocimiento dado. Dicha conclusión se parecerá más al consecuente de la regla original cuanto mayor sea el grado de cumplimiento del hecho con el antecedente de la regla (ver figura 3.1, página 49). Este razonamiento se resume principalmente en la generalización de la regla del Modus Ponens del razonamiento clásico.

2.5 Razonamiento Revisable

Aquí agrupamos un conjunto de técnicas eficaces de razonamiento cuando no se dispone de un modelo del mundo completo, consistente y constante. Los comienzos del razonamiento no monótono datan de principios de los 70 y su prehistoria puede ser incluso enmarcada en los 50, con el trabajo de McCarthy *Programs with Common Sense* ([**Luger, 1995**]) relacionado con el deseo de realizar razonamiento de *sentido común* en Inteligencia Artificial. Principalmente durante el período 1975-1979 se produjo la formalización del campo del razonamiento no monótono tal como lo conocemos actualmente, obteniendo su impulso definitivo en 1980 tras la publicación de una edición de *Artificial Intelligence Journal* dedicada exclusivamente al razonamiento no monótono. Desde entonces han aparecido gran cantidad de artículos describiendo aproximaciones al razonamiento no-monótono como extensiones de los sistemas lógicos: circunscripción, razonamiento por defecto, lógica modal para el manejo de la no-monotonicidad, etc. El libro ([**Brewka *et al.*, 1997**]) recoge una panorámica conjunta de estas técnicas.

Los sistemas de lógica de predicados de primer orden son *monótonos* en el sentido de que conforme se añaden nuevos axiomas, aparecen nuevas fórmulas bien formadas, pero nunca se vuelven inválidos los resultados anteriores. De acuerdo con el *principio de monotonía* si una conclusión Q puede ser obtenida de un conjunto dado de premisas P,

siempre podremos seguir deduciéndola a partir del conjunto de premisas resultante de añadir a P cualquier otra información. Pero, el razonamiento humano no trabaja de esta manera, y la obtención de nueva información puede llevarnos a cambiar creencias anteriores. El razonamiento humano es no monótono en el sentido de que partimos de una base con información parcial y cuando obtenemos más información es cuando alcanzamos las conclusiones.

Debemos admitir que nuestra capacidad para cambiar de opinión cuando las circunstancias lo exigen constituye un elemento crucial del sentido común. El objetivo básico del razonamiento no monótono es desarrollar modelos de sistemas de razonamiento similares al modo en que el sentido común es usado por los humanos. Razonamos guiándonos por el conocimiento obtenido a través de las experiencias anteriores, pero a la vez, permitiendo que esa base de conocimiento pueda ser variada si los hechos nos demuestran lo contrario. Por tanto, debe ser capaz de obtener conclusiones sin una garantía total, y ser suficientemente robusto para que cuando se compruebe que una conclusión alcanzada por razonamiento no monótono es errónea, se proceda a su revisión. Por eso también se le conoce como *razonamiento revisable.*

Un ejemplo clásico es que aceptamos que "todos los pájaros vuelan" y si sabemos que "Piolín es un pájaro", normalmente solemos asumir (en ausencia de evidencias de lo contrario) que "Piolín vuela". Sin embargo, en el conocimiento humano no existen tantas afirmaciones universales como aparecen en el lenguaje usual, y así, al tener nuevo conocimiento sobre que "Piolín es un pingüino", debemos retractarnos de nuestra anterior conclusión de que "Piolín vuela" ya que sabemos que los pingüinos no vuelan.

2.5.1 Razonamiento por Defecto

Se quiere usar el razonamiento no monótono para llevar a cabo lo que comúnmente se denomina *razonamiento por defecto* ("default reasoning"). Se pretende llegar a unas conclusiones basadas en lo que es más probable que sea cierto. Se han propuesto dos enfoques para lograrlo: la lógica no monótona y la lógica por defecto[1]. También estudiaremos un tipo de razonamiento no monótono muy común y que puede definirse en estas lógicas, la abducción.

[1]No debemos confundir la nomenclatura. Se utilizan los términos "razonamiento no monótono" y "razonamiento por defecto" para describir de forma genérica un tipo de razonamiento. Los términos "Lógica no monótona" y "Lógica por defecto" se usan por otra parte, para referirse a teorías formales concretas.

Lógica No Monótona

La *Lógica No Monótona* (NML, "Non-Monotonic Logic") descrita por McDermott y Doyle en 1980 ([**McDermott and Doyle, 1980**]) es un sistema que proporciona una base para razonar por omisión, en donde el lenguaje de la lógica de primer orden se aumenta con un operador modal *M*, que se lee como "es consistente".

Así, por ejemplo la sentencia "para todo par de personas, si son parientes y si el hecho de que uno se haya puesto de acuerdo con el otro es consistente con el resto de las suposiciones, entonces se concluye que se defenderán" quedaría:

$$\forall x \forall y (\text{Parientes}(x,y) \wedge M\ \text{EstáDeAcuerdo}(x,y) \rightarrow \text{Defenderá}(x,y))$$

Una vez que se ha aumentado la teoría para permitir sentencias de este tipo, debe resolverse un importante aspecto si se desea que la teoría sea igualmente semidecidible. Es necesario definir el concepto de "consistencia". Como en este sistema al igual que en la lógica de primer orden la consistencia es indecidible, necesitamos algún tipo de aproximación. Surge además un segundo problema que consiste en qué hacer cuando múltiples sentencias no monótonas, tomadas por separado, sugieren algunas formas de aumentar el conocimiento; sentencias, que si se toman todas juntas, resultan inconsistentes. En la formulación original de la NML, la semántica del operador modal *M*, que era autorreferencial, no estaba clara. Un sistema más reciente y muy similar, la *lógica autoepistémica* (AEL, "Autoepistemic Logic", [**Moore, 1985**]), resuelve algunos de estos problemas.

Lógica por Defecto

La *Lógica por Defecto* (DL, "Default Logic") de Reiter (1980) ([**Reiter, 1980**]) es una lógica alternativa para llevar a cabo un razonamiento basado en omisiones en la que se introduce un nuevo tipo de reglas de inferencia, las *reglas de inferencia por defecto*. Este enfoque determina reglas de inferencia de la forma:

Si *A* es verdadero y asumir *B* es consistente con la información existente, entonces se concluye que *C* es verdadero.

A es la *premisa*, *B* la *justificación* y *C* la *conclusión* de la regla por defecto. La segunda premisa "B es consistente con la información existente" es una declaración sobre la teoría y, por tanto, de orden superior ya que requiere evaluar el metapredicado *M B*, lo que en general no es sencillo. Las reglas por defecto pueden ser vistas como extensiones del conocimiento que tenemos del mundo, consiguiendo una completitud en ellas. Una *extensión* es el conjunto máximo de conclusiones tomadas desde una teoría por defecto. Una teoría por defecto se define como un par (D, P), donde D es un conjunto de reglas por defecto cerradas y P un conjunto de sentencias de primer orden. Hay que destacar que no todas las teorías por defecto tienen necesariamente extensiones. El problema de encontrar extensiones para el caso de teorías por defecto, aunque éstas sean sin prerrequisitos y a lo sumo dos literales por defecto, será NP-completo.

Abducción

La lógica estándar lleva a cabo deducciones de forma que dados dos axiomas $A \rightarrow B$ y A, entonces podemos deducir B. Sin embargo, si lo que tenemos es $A \rightarrow B$ y B, entonces no podemos concluir B ya que no está permitida por las reglas de la lógica estándar (ver página 23). De todas formas, aunque puede ser incorrecto, es posible que sea la mejor suposición que pueda hacerse. La derivación de conclusiones de esta forma es otra manera de razonamiento por defecto. Denominamos a este método *razonamiento por abducción* [2]. El proceso de razonamiento por abducción puede describirse con más precisión de la siguiente forma :

Dadas dos fbf , $A \rightarrow B$ y B, para cualquier expresión A y B, si es consistente asumir A, hacerlo.

Así por ejemplo, si el axioma dado es que "si se tiene sarampión entonces aparecen manchas rojas" y supongamos que lo que se observa son las manchas rojas, podría ser bueno concluir que se puede tener sarampión. El razonamiento por abducción es particularmente útil en muchos dominios si se asigna alguna medida a las expresiones resultantes. Estas medidas de certeza cuantifican el peligro de que el razonamiento por

[2] **Abducción**: silogismo cuya premisa mayor es evidente y la menor menos evidente o sólo probable. (*Diccionario de la Lengua Española.* Real Academia Española, Madrid, 1992).

abducción sea incorrecto, lo cual sucederá cuando existan otros antecedentes además de A que podrían producir B.

El razonamiento abductivo no es un tipo de lógica del estilo de la DL y la NML. En realidad, puede describirse sobre cualquiera de ellas. Sin embargo, se ha mencionado aquí explícitamente porque representa un tipo de razonamiento no monótono muy útil.

2.5.2 Razonamiento Minimalista

En este subapartado mostramos métodos para referirnos a un tipo muy específico y útil de cosas que son ciertas en general. Estos métodos se basan en alguna variante de la idea de *modelo mínimo*. Recordemos que un modelo de un conjunto de fórmulas es una interpretación que las satisface a todas, es decir, que las hace verdaderas. Aunque existen algunas definiciones diferentes sobre qué constituye un modelo mínimo, para nuestro propósito se dirá que un modelo es mínimo si no existen otros modelos en los que sean ciertas menos cosas. La idea que hay detrás del uso de modelos mínimos como base para el razonamiento no monótono sobre el mundo es la siguiente:

Existen muchas menos sentencias ciertas que falsas. Si algo es relevante y cierto, tiene sentido asumir que pertenece a nuestra base de conocimiento. Por lo tanto asumamos que las únicas sentencias ciertas son aquellas que necesariamente deben ser ciertas para que se mantenga la consistencia de la base de conocimiento.

Como ejemplo, supongamos que tenemos una mascota llamada "Orejotas". Como se trata de un perro añadiremos a nuestra base de conocimiento el hecho (predicado) `perro(orejotas)`, pero no es necesario añadir que "Orejotas" no es un gato, ni un caballo, ni un canario, etc., por tanto no aparecerán en la base de conocimiento los predicados `¬gato(orejotas)`, `¬caballo(orejotas)`, `¬canario(orejotas)`, ...

Por todo ello, ante un objetivo que no se puede demostrar a partir de nuestra base de conocimiento, nos podemos plantear dos tipos de respuesta:

- No sabemos si ese objetivo será cierto o falso puesto que no tenemos información suficiente sobre el mismo.

- Como no se puede deducir de nuestra base de conocimiento asumimos que es falso, por lo que se contesta negativamente a la consulta.

Veamos algunas propuestas formales que se basan en esta segunda idea.

Negación por fallo

Un tipo de razonamiento basado en esta idea es el concepto de PROLOG de negación como fallo, el cual proporciona una implementación de la noción para los sistemas basados en cláusulas de Horn. Un sistema raramente tiene a su disposición toda las información que sería necesaria para obtener una respuesta de la que se pueda asegurar que es correcta. Pero con frecuencia, cuando falta dicha información, pueden hacerse algunas suposiciones sensatas mientras no se presente evidencia contradictoria. La propiedad más acusada de este razonamiento es la de poder revisar sus conclusiones.

La programación lógica no usa la negación lógica clásica, sino un operador no monótono referido en la literatura técnica como *negación por defecto* o *por fallo*:

Si se plantea una pregunta P, se intenta deducir P de la base de conocimiento; si esto no es posible, se afirma *no* P.

La no monotonicidad de este operador nos permite ver a los programas lógicos como una teoría no monótona especial. Así, aunque los programas lógicos constituyen una subclase de la clase de todas las teorías no monótonas, son lo suficientemente expresivos para permitir la formalización de muchos e importantes razonamientos no monótonos, proporcionando cuestionarios fiables para nuevas formalizaciones. Pueden usarse como mecanismos de inferencia para formalismos no monótonos. El problema de encontrar mecanismos de inferencia eficientes, capaces de modelar el razonamiento humano es una de los mayores problemas de implementación e investigaciones en IA. Los últimos años han traído un rápido crecimiento de investigaciones que abordan las bases formales del razonamiento no monótono y la programación lógica, lo cual, sin embargo, no ha llevado emparejado un progreso similar en la implementación y desarrollo de herramientas de aplicación basadas en estas formalizaciones. Una de las principales causas de esta situación es el hecho de que la mayoría de los formalismos no monótonos son, en general, no tratables.

Suposición de un Mundo Cerrado

La suposición de un mundo cerrado (CWA, "Closed World Assumption") de Reiter (1978) sugiere una sencilla forma de razonamiento minimalista. La CWA dice que los únicos objetos que satisfacen un predicado P son aquellos que deben hacerlo. La CWA es particularmente poderosa como principio para razonar con bases de datos, asumiendo, por tanto, que son completas con respecto a las propiedades que describen.

Por ejemplo, se puede asumir sin peligro alguno que una base de datos sobre personal puede listar todos los empleados de una empresa. Si alguien pregunta si Martínez trabaja para la empresa, se puede responder no a no ser que aparezca explícitamente en la lista como un empleado. De forma similar, en una base de datos de una compañía aérea puede asumirse que contiene una lista completa de todas las rutas a las que vuela la compañía. Así, si se pregunta si hay un vuelo directo desde El Altet hasta Pekín, la respuesta debería ser no si no se encuentra nada en la base de datos.

Aunque la CWA es a la vez sencilla y poderosa, puede dar errores en la generación de respuestas apropiadas por dos razones :

- La primera es que esta suposición no es siempre cierta en el mundo; algunas partes del mundo no son realmente posibles de cerrar.
- La CWA producirá resultados apropiados exactamente en la misma medida en que sea cierta la suposición de que todos los hechos positivos relevantes están en la base de conocimiento.
- El segundo problema que atormenta a la CWA surge del hecho de que es un proceso de razonamiento puramente sintáctico. Así, como podría esperarse, sus resultados dependen de la forma en que se proporcionan las afirmaciones.

Circunscripción

Aunque la CWA captura parte de la idea de que algo que no debe ser necesariamente cierto debería ser asumido como falso, no la captura completamente. Posee dos limitaciones esenciales:

1. Opera sobre predicados individuales sin considerar las interacciones entre los predicados definidos en la base de conocimiento.
2. Asume que todos los predicados tienen listadas todas sus instancias. Aunque esto sea cierto en muchas bases de datos, en bastantes bases de conocimiento no lo es. Puede asumirse razonablemente

que algunos predicados están completamente definidos (es decir, la parte del mundo que definen es cerrada), pero otros no (es decir, la parte del mundo que describen es abierta).

Para manipular estos problemas, se han propuesto distintas teorías sobre la *circunscripción* ([**McCarthy, 1980**]). John McCarthy, con esta técnica, intenta tratar las excepciones a una regla general. A menudo la gente acepta y usa reglas universales incompletas como "(todos) los pájaros vuelan", principio que tiene excepciones. Nosotros usamos estas reglas, quizá incluso inconscientemente, a menos que tengamos una razón para pensar que tratamos con un caso donde la regla no se aplica. Normalmente no listamos todas las excepciones como parte de la regla, y comprendemos que pueden haber muchas excepciones que no conocemos. Estas excepciones, en principio pueden ser potencialmente infinitas, por lo que su idea consiste en circunscribir [3] (delimitar) el conjunto de las posibles excepciones de un enunciado. En las distintas teorías basadas en la circunscripción se añaden nuevos axiomas a la base de conocimiento existente. El efecto de estos axiomas consiste en forzar una interpretación mínima sobre una parte seleccionada de la base de conocimiento. En particular, cada axioma específico describe una forma de delimitar (es decir, de circunscribir) el conjunto de valores para los que un axioma particular de la teoría original sea cierto.

2.5.3 Sistemas de Mantenimiento de la Consistencia

La idea de un *Sistema de Mantenimiento de la Consistencia* (TMS, "Truth Maintenance System") descrita por Doyle en 1979 ([**Doyle, 1979**]), surge como una forma de proporcionar la habilidad de trabajar con una vuelta atrás dirigida por dependencias o justificaciones para poder soportar el razonamiento no monótono. Así, de forma general, podemos definir un TMS como un sistema que mantiene un conjunto de hipótesis y justificaciones de forma que es capaz de responder sobre la validez de una o varias afirmaciones en el contexto sobre el que entiende y es capaz de actualizar sus creencias en función de las modificaciones aportadas. Sin embargo, el volumen de computación exigido tiende a crecer exponencialmente a medida que crece la base de conocimiento, y la cantidad de hechos que se necesitan para llevar a cabo un auténtico razonamiento de sentido común acabarían dejando atascado al TMS. Se han ido desarrollado investigaciones que exploran distintas técnicas para abordar un sistema de mantenimiento de la verdad. Un TMS permite conectar las

[3] **Circunscribir** : reducir a ciertos limites o términos alguna cosas. (*Diccionario de la Lengua Española.* Real Academia Española, Madrid, 1992).

Volver al índice/Tornar a l'índex

aserciones mediante una red de dependencias. Veamos algunos sistemas existentes en la literatura sobre el tema:

- Un sistema de mantenimiento de la verdad *basado en justificaciones* (JTMS) no conoce nada sobre la estructura de las aserciones en sí mismas y trata los nodos de la red como átomos, lo cual significa que no hay relaciones entre ellos excepto aquellas que se sitúan explícitamente en las justificaciones. El único papel de los TMS es el de servir como libro de anotaciones para un sistema de resolución de problemas, que a su vez le proporciona tanto las aserciones como las dependencias entre las aserciones.
- Un sistema de mantenimiento de la verdad *basado en la lógica* (LTMS) puede detectar contradicciones de forma automática.
- Los sistemas de mantenimiento de la verdad *basados en suposiciones* (ATMS) mantienen en paralelo varios caminos alternativos en distintos contextos, a los cuales corresponde un conjunto de suposiciones consistentes. Al evolucionar el razonamiento, el universo de contextos consistentes va podándose conforme se detectan contradicciones.

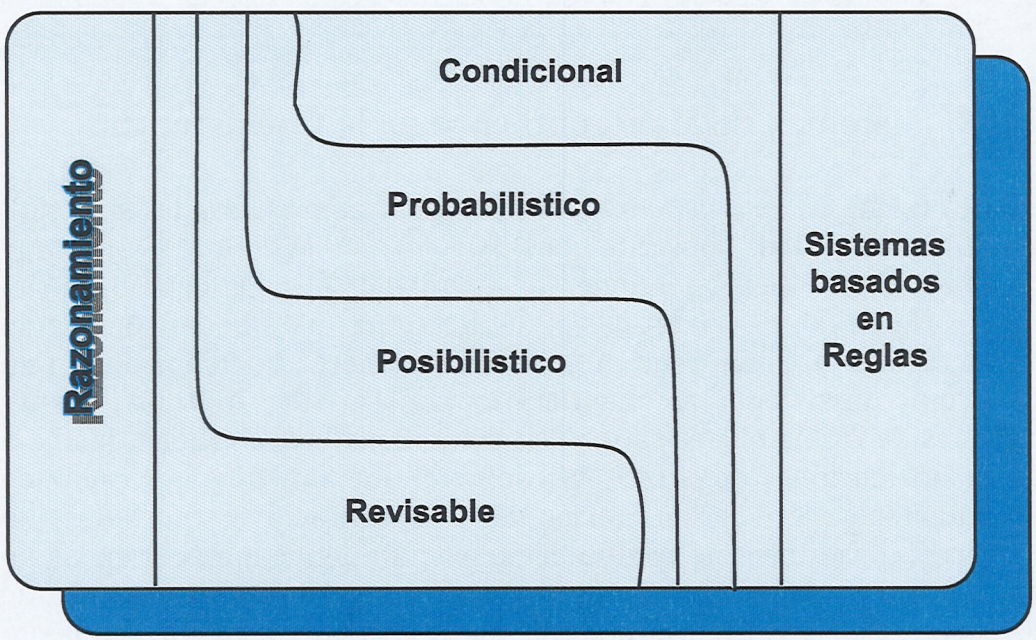


Figura 2.1: Esquema de Sistemas de Razonamiento

Capítulo 3

Análisis y Caracterización de los Sistemas Basados en Conocimiento

> Cuando pensamos en la perfección evolutiva de otros organismos, el hombre parece, por supuesto, una criatura desgarbada y pobremente constituida. Como organismo, apenas puede competir con las criaturas que ocupan cualquier nicho ecológico específico de la Tierra. No obstante, ha conseguido dominar este planeta gracias únicamente a una especialización bastante importante: su cerebro.
>
> *Nueva guía de la Ciencia.* Isaac Asimov

En los primeros años las tareas inteligentes que se abordaban con ordenador eran de carácter lógico-formal (juego de ajedrez, demostración automática de teoremas, micromundos, etc.). El siguiente paso fue simular, además de la capacidad abstracta de deducir, el saber real de un individuo humano considerado como experto o especialista en determinadas áreas o dominios (médicos, ingenieros, etc.) ([**Dym and Levitt, 1991**]). Este saber se compone de varios ingredientes:

- la lógica
- la información concreta sobre un dominio de la realidad

- una serie de reglas prácticas que uno va aprendiendo con la experiencia

Los sistemas basados en conocimiento pretenden formalizar estos aspectos para introducirlos en los sistemas informáticos y, así, automatizar esas tareas específicas de un dominio dado. Para una descripción básica de los sistemas expertos basados en conocimiento, principios, diseño y programación, se puede consultar [**Davis and Lenat, 1982**], [**Schreiber *et al.*, 1993**], [**Durkin, 1994**] y [**Giarratano and Riley, 1994**], entre otros.

3.1 Introducción

Los sistemas basados en conocimiento son a menudo llamados también sistemas expertos ([**Feigenbaum *et al.*, 1988**]). Podríamos definir a un *sistema experto* como un programa inteligente que utiliza conocimiento especializado y un mecanismo de inferencia para resolver problemas de un determinado dominio de forma análoga a como lo haría un experto humano. En una situación ideal, un sistema experto se comportaría de la misma forma que lo haría un experto humano en la materia, con las siguientes ventajas:

- *Permanencia*: la duración de un sistema experto es ilimitada, su experiencia es permanente y no se cansa.
- *Economía*: una vez construido el sistema experto no presenta ningún coste adicional, salvo el de mantenimiento de la base de conocimiento; además, el conocimiento que contiene es fácil de actualizar.
- *Complejidad*: es capaz de utilizar grandes cantidades de conocimiento y, además, podemos combinar el conocimiento de varios expertos humanos en un solo sistema.
- *Reproducción*: son fáciles de duplicar, de forma que se puede disponer de uno en cualquier lugar que sea necesario.
- *Fiabilidad*: son consistentes.
- *Facilidad de uso*: son documentables.
- *Velocidad*: en general, los sistemas expertos pueden responder a las preguntas y resolver problemas mucho más rápidamente que un experto humano.

A lo largo de la década de los 70 se desarrollaron en universidades y centros de investigación diversos sistemas experimentales, entre los que podemos destacar DENDRAL (sistema experto para la interpretación y propuesta de ensayos para reconocer compuestos químicos), MYCIN (sistema experto para el diagnóstico de enfermedades infecciosas, [**Shortliffe, 1976**]), INTERNIST (sistema experto para el diagnóstico de enfermedades internas) y PROSPECTOR (sistema experto para la evaluación de potenciales yacimientos mineros, [**Duda *et al.*, 1977**]).

3.1.1 Tipos de Sistemas Basados en Conocimiento

Podemos clasificar los sistemas expertos por distintos conceptos. Uno de ellos es por el tipo de tarea que realizan. Aplicando este criterio podemos considerar los tipos que aparecen a continuación, aunque los sistemas expertos reales combinan varias de estas características ya que algunas son complementarias :

Sistemas de Control : gobiernan el comportamiento del sistema para que cumpla determinadas especificaciones. Por ejemplo el control del tratamiento de un paciente en un hospital.

Sistemas de Diseño : construyen y configuran objetos siguiendo un conjunto de especificaciones y restricciones del problema. Como por ejemplo el diseño de un sistema informático a partir de las necesidades definidas por el usuario (memoria, velocidad, ...).

Sistemas de Diagnóstico : a partir de determinadas observaciones (síntomas) pueden deducir un fallo o mal funcionamiento del sistema. Aquí tendríamos, por ejemplo, a los sistemas de diagnóstico para localizar los posibles averías en un circuito electrónico a partir de los resultados de un test.

Sistemas de Instrucción : sistemas que guían el aprendizaje de un estudiante en un determinado tema, analizando las respuestas del estudiante y corrigiendo los fallos.

Sistemas de Interpretación : a partir de determinadas observaciones deducen la descripción de la situación intentando entenderla o clasificarla. Así funcionan, por ejemplo, los sistemas de análisis de imágenes.

Sistemas de Monitorización : comparan, de forma continua, los valores reales observados con los valores esperados de forma que puedan activar señales de alarma si se detecta un mal funcionamiento. Tendríamos, por ejemplo, los sistemas para monitorización de pacientes en medicina.

Sistemas de Planificación : diseñan un plan de acciones para conseguir un objetivo dado. Un ejemplo podría ser la planificación de las diferentes tareas a realizar por un robot para conseguir un determinado objetivo.

Sistemas de Predicción : deducen posibles consecuencias a partir de las situaciones dadas. Por ejemplo, un sistema experto para predecir la evolución de cosechas agrícolas que permita predecir el daño esperado en una cosecha ante la invasión de una plaga de insectos.

Sistemas de Prescripción : recomiendan soluciones para enmendar el mal funcionamiento de un sistema. Un sistema experto que recomienda una determinada terapia para los pacientes que sufren una depresión pertenecería e este tipo ([**Pujol *et al.*, 2001**]).

Sistemas de Selección : identifica la mejor elección posible a partir de una lista de posibilidades. Por ejemplo un sistema experto que asiste en la selección de robots industriales en un determinado entorno de trabajo.

Sistemas de Simulación : modelan interacciones entre componentes del sistema para estudiarlo bajo diversas condiciones. De todos son conocidos ejemplos de simuladores.

Otra forma de clasificar los sistema expertos es según la naturaleza del problema con el que se enfrentan, de forma que los podemos dividir en:

- Sistemas que tratan con problemas esencialmente *deterministas*, que pueden ser formulados mediante reglas y se obtienen las conclusiones con razonamientos lógicos.
- Sistemas que se enfrentan con problemas de naturaleza *estocástica*, con situaciones inciertas a los cuales debemos incorporar algún mecanismo para tratar la incertidumbre.

3.1.2 Sistemas Expertos Basados en Reglas

El tipo de sistemas expertos que más se ha estudiado de forma teórica y formal ha sido el de los *Sistemas Expertos Basados en Reglas de Producción*[1], es decir, aquellos sistemas que utilizan reglas

[1]**Producción**: término usado en la psicología cognitiva para describir la relación entre situaciones y acciones. La estructura básica de una producción consiste en un *antecedente* que describe la situación y un *consecuente* que describe alguna acción a emprender si se da la situación.

de producción como formalismos de representación del conocimiento ([**Buchanan and Shortliffe, 1984**]). La idea de utilizar reglas de producción tiene sus antecedentes en el trabajo de Newell y Simon ([**Newell and Simon, 1972**]) que usaban estas reglas para modelar la capacidad humana de resolver problemas. Además, es el tipo de sistema experto más fácil de comprender y, por tanto, el mejor para empezar a estudiar y desarrollar sistemas expertos. Podríamos definir un sistema experto basado en reglas como un programa de ordenador que procesa información específica de un problema contenida en la base de hechos con un conjunto de reglas contenidas en la base de conocimiento, usando un motor de inferencia para deducir nueva información.

Tal como se vio anteriormente, podemos definir una regla de producción como un par condición/acción cuya estructura es

Si *condición* **Entonces** *acción*

Esto significa que en caso de que se satisfaga la condición (o combinación lógica de condiciones elementales) se debe realizar la acción (o combinación lógica de acciones elementales).

Los sistemas basados en reglas ofrecen las siguientes ventajas:

- *Modularidad*, ya que cada regla de producción es un pequeño elemento de conocimiento independiente y, por tanto, puede ser verificada independientemente cada pieza de conocimiento.
- *Modificabilidad*, ya que, como consecuencia de la modularidad, se puede modificar, eliminar o añadir fácilmente reglas a la base de conocimiento.
- *Naturalidad*, ya que los humanos expresamos el conocimiento como sentencias condicionales (si ... entonces), las reglas de producción son formas naturales de expresar conocimiento.
- *Separación entre Control y Conocimiento*, ya que separan el conocimiento contenido en la base de conocimiento de los procedimientos de control realizados por el motor de inferencia.
- *Incrementalidad*, ya que como cada regla es una pieza independiente de conocimiento, el incremento del número de reglas del sistema nos proporcionará un aumento gradual en el conocimiento del sistema acerca del problema.
- *Uso del Conocimiento Relevante*, ya que el sistema en cada inferencia solamente utilizará las reglas que sean relevantes para el problema tratado.

- *Transparencia*, es decir, la posibilidad de obtener una explicación de las decisiones y de la solución, ya que podemos obtener una traza de las reglas utilizadas que nos proporcionarán una descripción de cómo se ha hecho el razonamiento.
- *Consistencia*, ya que la rígida estructura de las reglas nos permite comprobar la consistencia del sistema.
- *Utilización de Heurísticas*, ya que podemos incorporar las heurísticas o trucos aprendidos de la experiencia de un experto humano al sistema de reglas.

Como principales inconveniente que presentan estos sistemas podemos enumerar:

- Requieren *emparejamiento preciso*, ya que el sistema intenta emparejar los antecedentes de las reglas con los hechos de la memoria de trabajo. Esto se puede evitar con técnicas de razonamiento aproximado.
- Puede contener *relaciones opacas* entre reglas, ya que podemos tener reglas encadenadas lógicamente, que pueden no encontrarse juntas y por tanto ser difíciles de localizar.
- *Lentitud*, ya que cuando la base de conocimiento contiene un número muy elevado de reglas estos sistemas pueden llegar a ser ineficientes.
- Son *inapropiados* para cierto tipo de problemas, cuando el conocimiento en ese dominio no pueda ser capturado por reglas.

En la actualidad el diseño de sistemas expertos suele llevarse a cabo con la ayuda de herramientas comerciales que ofrecen facilidades para modelar el conocimiento del dominio mediante reglas. Dentro de este tipo de sistemas basados en reglas podemos enmarcar CLIPS (al ser la herramienta que utilizamos, hacemos una breve descripción en el apartado 5.2.2). El lenguaje de programación lógica Prolog también facilita enormemente el trabajo con información simbólica.

3.2 Estructura de un Sistema Basado en Conocimiento

La complejidad de las tareas de un sistema experto hace que lo estructuremos en distintos módulos. La estructura básica de un sistema experto se compondría de :

Base de Conocimiento : parte del sistema experto que contiene el conocimiento relativo al dominio específico del problema sobre el que trata dicho sistema experto.

Base de Hechos o Memoria de Trabajo : parte del sistema experto que contiene las circunstancias particulares del problema de la sesión de trabajo, bien proporcionados por el usuario o bien inferidos por el propio sistema. También se le llama *contexto de la sesión.*

Motor de Inferencia : parte del sistema experto que confronta los hechos particulares de la Memoria de Trabajo con el conocimiento contenido en la Base de Conocimiento para obtener conclusiones acerca del problema. Coordina la información procedente de los distintos módulos.

Interfaz de Usuario : parte del sistema experto que solicita y/o ofrece información al usuario.

Si hay una realimentación desde el motor de inferencia a la base de conocimiento, entonces estamos ante la capacidad de *aprendizaje* del sistema experto. Estos distintos módulos que conforman la estructura básica de un sistema experto genérico serán desarrollados con más detenimientos en los siguientes apartados.

3.2.1 Bases de Conocimiento

Como se puede desprender de lo dicho hasta el momento, en los sistemas expertos basados en reglas de producción la base de conocimiento está formada por un conjunto de reglas, que reflejan el conocimiento del experto en la resolución de ese determinado tipo de problemas ([**Anderson, 1993**]).

Podemos diferenciar distintos tipos de reglas según el tipo de conocimiento que representen:

- Conocimiento sobre el dominio: representan conocimiento acerca de cómo aproximarse a la solución de un problema en función de las circunstancias particulares del momento (relaciones, recomendaciones, directivas, estrategias y heurísticas).
- Conocimiento sobre cómo usar el conocimiento: a menudo al resolver un determinado problema el experto utiliza su experiencia para dirigir el proceso de resolución del mismo. Este conocimiento es diferente del comentado en el punto anterior ya que no caracteriza al dominio. Este tipo de conocimiento lo llamamos *metaconocimiento* puesto que utiliza el conocimiento que tenemos sobre

el dominio para determinar cómo encontrar la mejor solución al problema estableciendo estrategias de control. Este conocimiento lo representamos con *metareglas*, es decir, reglas que describen cómo deben ser usadas otras reglas, proporcionando directrices al sistema experto durante la sesión de trabajo.

3.2.2 Motor de Inferencia

Una vez que tenemos representado el conocimiento, necesitamos un procedimiento de razonamiento que nos permita extraer conclusiones que se deriven de la base de conocimiento y de la base de hechos. Este procedimiento de razonamiento vendrá dado por el motor de inferencia. Así, en un sistema basado en reglas, el motor de inferencia deberá ser capaz de realizar la búsqueda de las reglas pertenecientes a la base de conocimiento que pueden ser disparadas de acuerdo con la información disponible en la base de hechos. Así, esa regla será *activada*. Diremos que una regla es *disparada* cuando, al satisfacerse la condición de la regla, ejecutamos la acción asociada a la misma.

El motor de inferencia se encargará de seleccionar, de alguna forma, las reglas candidatas, es decir, aquellas que pueden producir algún cambio en la base de hechos al satisfacerse las condiciones de dicha regla. Esta búsqueda puede hacerse de distintas maneras, aunque las formas más sencillas son el encadenamiento hacia delante y el encadenamiento hacia atrás. Llamamos *encadenamiento* porque se enlazan sucesivamente las reglas al satisfacerse las condiciones de una como consecuencia de las acciones de otra.

La elección de un tipo de encadenamiento u otro vendrá determinada por el problema a resolver. Es conveniente que el estilo de búsqueda utilizado por el sistema experto sea, desde el punto de vista semántico, el más similar al utilizado en el razonamiento humano, es decir, se realice en la misma secuencia que un humano encontraría la solución en el campo de aplicación en el que se trate. Esto es importante, sobre todo, cuando durante el proceso de razonamiento existe alguna interacción con el usuario y se quiere que el proceso sea transparente para él (aportándonos las explicaciones pertinentes).

Encadenamiento hacia atrás

Al *encadenamiento hacia atrás* también se le conoce como *guiado por objetivos* ya que el sistema empieza con el objetivo, o hecho que se intenta demostrar, e intenta verificar si se satisface o no. De esta manera el sistema se centra exclusivamente en las reglas que sean relevantes pa-

ra solucionar el problema planteado, es decir, aquellas que nos puedan llevar al hecho que estamos intentando confirmar. Así, vamos generando pasos de la cadena de inferencia que conectan los siguientes tipos de información:

objetivos	⇝	...	⇝	datos
hipótesis	⇝	...	⇝	evidencias
explicaciones	⇝	...	⇝	observaciones
conclusiones	⇝	...	⇝	hechos

Cuando el motor de inferencia trata de verificar un objetivo, en primer lugar identificará todas las reglas que, de ser disparadas, permitan concluir que el objetivo es cierto. Posteriormente, intentará disparar una por una las reglas seleccionadas, y cómo para ello deberán satisfacerse las condiciones correspondientes, éstas se convertirán en nuevos objetivos que deben, a su vez, ser verificados. De esta forma, de manera recursiva, se intentarán verificar objetivos, terminando bien cuando se consiga saber si ese objetivo es satisfecho o no, o bien cuando no queden reglas que puedan ser disparadas.

La principal ventaja del encadenamiento hacia atrás es que únicamente trabaja con los datos y reglas que estén relacionados con el problema concreto que está intentando solucionar. Este método será adecuado cuando queremos utilizar el sistema experto para comprobar una hipótesis concreta.

El encadenamiento hacia atrás no tiene porque ser siempre igual y, por tanto, existen diferentes estrategias de selección de la regla a disparar de entre todas las candidatas. Las dos estrategias más utilizadas son la *búsqueda en profundidad* y la *búsqueda en amplitud.* También podemos hablar de las estrategias que determinan en qué orden se consideran las reglas. Para ello existen, también, distintas posibilidades:

1. Considerar el orden en el que aparecen en la base de conocimiento.
2. Asignar a cada regla una *prioridad.*
3. Dar preferencia a aquellas reglas cuya condición sea menos compleja.
4. Aleatoriamente

Encadenamiento hacia delante

El *encadenamiento hacia delante* es también conocido como *guiado por los datos* porque utiliza la información suministrada por el usuario para

buscar las reglas e inferir las consecuencias que se deriven de ellas. Partimos de los hechos contenidos en la memoria de trabajo introducidos al principio de la consulta por el usuario y el sistema experto va añadiendo hechos que se vayan deduciendo debido a que existe una regla en la base de conocimiento cuyo consecuente es el referido hecho y, con la información que tiene el sistema, se satisface el antecedente de la regla. De forma complementaria a lo dicho en el apartado anterior, vamos generando pasos de la cadena de inferencia que conectan los siguientes tipos de información:

datos	⇝	...	⇝	objetivos
evidencias	⇝	...	⇝	hipótesis
observaciones	⇝	...	⇝	explicaciones
hechos	⇝	...	⇝	conclusiones

Al realizar una consulta, y una vez introducidos todos los hechos particulares del problema a tratar en la memoria de trabajo, el motor de inferencia recorre todas las reglas de la base de conocimiento hasta encontrar una que se pueda disparar. Cuando se dispara la regla, el conjunto de acciones del consecuente de dicha regla se incorporan a la base de hechos (memoria de trabajo), y se repite el proceso. Finalizaremos cuando no haya más reglas que puedan ser disparadas.

En este caso, la selección de la regla que se va a disparar entre todas las candidatas es irrelevante, ya que se van a disparar todas ellas. De esta forma, lo más sencillo es realizar un recorrido secuencial por la base de conocimiento y analizarlas en el orden en que aparecen en ella.

Existen también modelos que combinan ambos tipos de razonamiento (en ambas direcciones). Se podría utilizar un razonamiento mixto, que reflejara un compromiso entre ambos modos de encadenamiento. En una primera fase se aplicaría un razonamiento hacia delante que generaría un conjunto pequeño de objetivos inmediatos; en una fase de confirmación se encargaría de verificar y seleccionar los objetivos de la fase anterior. Además se le puede incorporar razonamiento aproximado, si para activar la regla no obligamos a que se cumpla estrictamente el antecedente. En este razonamiento la conclusión se parecerá al consecuente de la regla tanto como la premisa al antecedente de la regla. En la figura 3.1 vemos gráficamente estos tres procesos de inferencia: en la parte superior una inferencia hacia adelante estricta (modus ponens), en el medio una inferencia hacia atrás estricta (modus tollens) y en la parte inferior una inferencia aproximada.

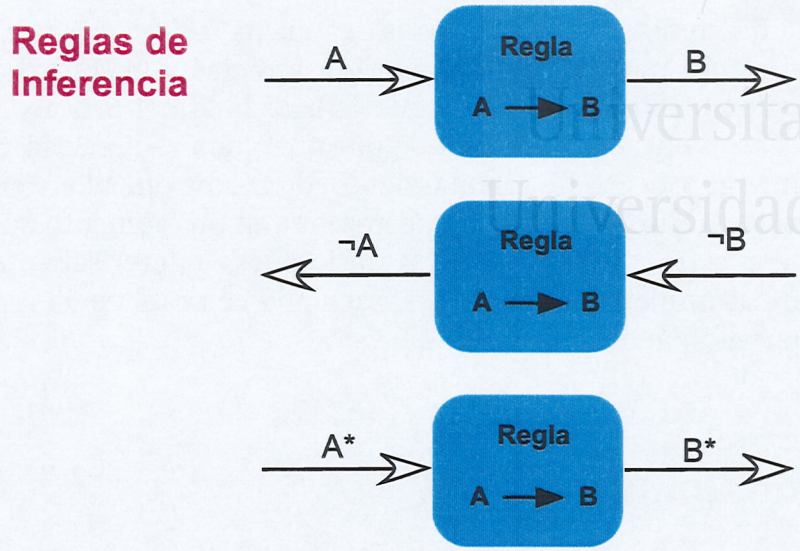


Figura 3.1: Distintos Procesos de Inferencia

3.2.3 Interfaz de Usuario

Este módulo de los sistemas expertos es el encargado de comunicarse con el usuario, proporcionando un dialogo natural, característico de este tipo de programas frente a los algorítmicos. Dicho interfaz debe ser bidireccional permitiendo tanto solicitar información al usuario (información que nos permitirá ir construyendo la *base de hechos*) como ofrecer explicaciones de las decisiones tomadas por el sistema experto.

Tal como hemos comentado antes al hablar de la propiedad de transparencia de los sistemas expertos, la forma de trabajar de los sistemas basados en reglas hace que sea muy sencillo implementar explicaciones que justifiquen sus conclusiones. En concreto, los sistemas expertos suelen incorporar explicaciones de dos tipos:

- Explicaciones del tipo *¿por qué?* ("why"): ¿Por qué estás interesado en esta información? Cada vez que el sistema pide al usuario la verdad o falsedad de un hecho, éste puede preguntar por qué le hace esa pregunta. El sistema, entonces, explicará la regla que quiere aplicar y que justifica el hecho que quiere comprobar.
- Explicaciones del tipo *¿cómo?* ("how"): ¿Cómo has obtenido esta conclusión? Cada vez que el sistema llega a una conclusión el usuario puede preguntarle cómo ha llegado hasta ella. El sistema responderá mostrándole el encadenamiento de reglas que le ha permitido obtener dicha conclusión.

Es fácil incorporar mecanismos para que un sistema experto genere explicaciones. La explicación del tipo ¿cómo? la podemos considerar

como una traza de cómo se ha obtenido la respuesta, es decir, el *árbol de prueba* de cómo la conclusión se sigue de las reglas y hechos de la base de conocimiento. Son por tanto requeridas al final del proceso de razonamiento. Las explicaciones del tipo ¿por qué? son requeridas por el usuario durante el proceso de razonamiento, de forma que el sistema experto solicita al usuario información que necesita en un momento dado para continuar con su razonamiento y el usuario puede querer saber por qué la necesita. Su implementación es tan sencilla como decir la regla que se está intentando activar.

3.3 Sentido Común

Los sistemas expertos funcionan bastante bien en dominios concretos, pero fallan al tener que trabajar con conocimiento general, en eso que llamamos sentido común ([**Davis, 1990**] y [**McCarthy and Lifschitz, 1990**]), y que algunos investigadores de IA han bautizado como *conocimiento ingenuo*. Lo primero sería saber qué es el *sentido común*. McCarthy ([**McCarthy, 1959**]) escribía que "podremos decir que un programa tiene sentido común si automáticamente deduce por sí mismo un conjunto suficientemente amplio de consecuencias inmediatas de cualquier cosa que se le diga y de lo que ya conoce". También decía que "nuestro objetivo final es hacer programas que aprendan de sus propias experiencias tal como efectivamente hacen los humanos". Se han entresacado estos párrafos porque se quiere hacer notar dos aspectos: uno es que cada uno de nosotros posee una gran cantidad de conocimiento que ha ido adquiriendo a lo largo de su vida; el otro es que dicho conocimiento los vamos aprendiendo minuto a minuto y en cada segundo que pasa derivados de nuestros propios actos.

El sentido común es el menos común de los sentidos

Los humanos sabemos un motón de cosas que vamos asimilando desde el momento en que nacemos: aprendemos a leer, escribir y calcular en nuestra infancia, estudiamos materias generales en la escuela, nos preparan en aspectos más específicos en la universidad y en nuestro trabajo diario seguimos aprendiendo. Pero además aprendemos de nuestros padres y familiares, cuando estamos con los amigos, de los libros que leemos, de las películas que vemos, de lo que sentimos, de nuestra interacción con el entorno, etc. Todo va configurando nuestro bagaje cultural, ese conocimiento implícito del mundo, lo que podríamos llamar inteligencia general. Además adquirimos conocimiento muy concreto y cerrado en sí mismo, en campos y ámbitos delimitados. Las máquinas actuales trabajan muy bien en contextos muy específicos, pero aún no se ha encontrado la fórmula para incorporar sentido o conocimiento general a cada objeto. Según [**Guha and Lenat, 1990**] "quizá, la verdad más

dura a la que enfrentarse, de la cual la IA ha intentado escapar durante 34 años, es que, probablemente, no hay una forma elegante, ni poco costosa, de obtener esta inmensa base de conocimiento".

El proyecto CYC[2] ([**CYC web, 2001**]), liderado por Douglas Lenat, pretende desarrollar una base de conocimiento de sentido común que nos permita realizar inferencias fácil y rápidamente. El proyecto CYC está configurado con una inmensa multicontextual base de conocimiento y un eficiente motor de inferencia. La base de conocimiento está construida sobre un núcleo de más de un millón de aserciones (reglas) diseñadas para capturar conocimiento acerca del mundo ([**Lenat and Guha, 1990**]).

[2]El nombre viene de la sílaba central de *encyclopedia*.

Volver al índice/Tornar a l'índex

Capítulo 4

Análisis y Caracterización de los Agentes Inteligentes y Sistemas Multiagente

> Si el organismo contiene en su cabeza un "modelo a pequeña escala" de la realidad externa y de todas sus posibles acciones, será capaz de probar diversas opciones, decidir cuál es la mejor, preparar su reacción ante situaciones futuras antes de que éstas surjan, emplear lo aprendido de experiencias pasadas en situaciones del presente y del futuro y, en toda situación, reaccionar ante los imponderables que tenga que enfrentar de manera satisfactoria, eficiente y sin riesgos.
>
> *The Nature of Explanation.* K. J. Craik

Las tendencias actuales hacen evidente que la complejidad del software continuará incrementándose dramáticamente en los próximos años. Cada día aparecen nuevos productos software, ofreciéndonos una amplia variedad de servicios. Pero mientras la información y los servicio permanezcan aislados perderemos valor. Cada vez es mayor la demanda de programas que puedan interactuar. La *Ingeniería del Software Basada en Agentes* ([**Shoham, 1993**]) fue desarrollada para facilitar la creación de software que interactúe, de forma que las aplicaciones son escritas como *Agentes de Software.* La naturaleza

distribuida y dinámica de los datos y las aplicaciones requieren que el software no sólo responda a las solicitudes de información de los usuarios, sino que se anticipe, adapte y busque activamente maneras de darle soporte. Los sistemas no sólo deben ayudar a administrar la cooperación entre sistemas distribuidos. En los últimos años muchos productos software de consumo contienen en su interior sistemas basados en agentes ([**Guilfoyle, 1995**]).

Como resultado de estas necesidades, los agentes surgen como área de rápido desarrollo donde se aúnan un gran número de disciplinas tan diversas como Interacción Humano-Ordenador, Ingeniería del Software, Redes, Sociología e Inteligencia Artificial.

Existe un gran número de actividades donde es posible aplicar la tecnología de agentes, entre las que se encuentran las interfaces (inteligentes) de usuario ([**Cousins, 1994**]), las telecomunicaciones, la administración de redes, los sistemas operativos ([**Etzioni *et al.*, 1993**]), el comercio electrónico y la organización y recolección de información ([**Barceinas, 1998**]). Ejemplos de misiones que pueden ser delegadas a los agentes son la filtración de mensajes de correo electrónico o de información de acuerdo a las preferencias del usuario, el manejo de procedimientos de administración rutinaria y la negociación con otros usuarios o sus agentes para fijar horas de reunión, entre muchas otras.

4.1 Introducción

4.1.1 Agentes Inteligentes

La evolución de la Inteligencia Artificial nos ha llevado desde los prometedores años sesenta, en los que se esperaba llegar a la creación de una máquina inteligente (una "supermáquina") por medio de técnicas de tratamiento formal de problemas generales bien definidos y estructurados, hasta la situación actual, mucho más realista , en la que los avances tecnológicos se centran en dotar a los sistemas de componentes con ciertas conductas que podríamos calificar como inteligentes, tratando problemas que impliquen conocimiento muy especifico. Podemos considerar que el conocimiento no está en un único sistema que todo lo sabe, sino que aparece distribuido en todo el sistema y que se enriquece de sus interacciones. Tal como plantea [**Rizo, 2001**], "la actualidad real de 2001, no la imaginada por Kubrick, desde el punto de vista de la IA, podemos plantearla como la etapa de los *componentes inteligentes artificiales*". Esto nos conduce a una visión más aplicada de la IA.

Investigadores en Inteligencia Artificial desean construir máquinas

(robots, agentes, programas, ...) que parezca que piensen, sientan y vivan ([**Langton, 1989**], [**Bates, 1994**], [**Franklin, 1995**] y [**Sloman,**]). Dotarles de cualidades humanas: razonamiento, capacidad de resolver problemas, capacidad de aprender, percepción y sentimientos. Pero el calificativo de inteligentes aplicado a nuestros agentes no va en la línea de que pasen el test de Turing (apartado A.3.3) sino más bien en que realizan tareas por nosotros, de forma más productiva, al permitirnos invertir menos tiempo y centrarnos en tareas más interesantes (complejas o importantes). Así, un agente exhibirá inteligencia si actúa de forma racional en la consecución de un objetivo. Según [**Maes, 1995**], los productos comerciales disponibles escasamente justifican el nombre de agentes y peor aún el adjetivo de inteligentes.

Distintos tópicos de Inteligencia Artificial (entre ellos los vistos de representación del conocimiento y razonamiento) se combinan para construir agentes inteligentes. La cantidad de inteligencia que nuestro agente demostrará en un determinado dominio y su capacidad de hacer cosas estará en función del tipo de representación del conocimiento que estemos usando y de la potencia de razonamiento. En los últimos años están apareciendo libros de Inteligencia Artificial que ofrece una visión unificada y coherente, con el hilo conductor de los agentes inteligentes, de su construcción y análisis como [**Russell and Norvig, 1996**] y [**Nilsson, 2001**]. Para un tratamiento básico sobre los agentes inteligentes, fundamentos teóricos y aplicaciones, se puede consultar [**Wooldridge and Jennings, 1995**], [**Wooldridge *et al.*, 1996**] y [**Brenner *et al.*, 1998**], entre otros.

4.1.2 Sistemas Mutiagente

Al igual que varias personas podemos colaborar para resolver un problema, podemos tener múltiples agentes trabajando juntos para conseguir un objetivo común. El problema aparece cuando dos seres inteligentes intentan comunicarse, ya que deben de hablar el mismo lenguaje, estar de acuerdo en el significado de los símbolos de ese lenguaje, tener un mecanismo de comunicación para el intercambio de mensajes, no pueden hablar ni escribir en la misma página al mismo tiempo, etc. Este apartado trata de Sistemas Multiagente e intenta desarrollar los aspectos mencionados. Los *agentes colaborativos* trabajan juntos como un equipo, combinando sus conocimientos y habilidades especializadas para resolver problemas grandes. La comunicación entre ellos se convierte, por tanto, en un aspecto clave.

Algunas características básicas de los sistemas multiagente son:

- Cada agente tiene una *vista limitada* del estado del mundo.
- *No* hay un *control global.*
- La información está *descentralizada.*
- La computación es *asíncrona.*

En un sistema multiagente, donde un conjunto de agentes (en general, distribuidos) trabajan para resolver un problema global, la metodología usada consta de tres pasos:

Divide y Vencerás

1. La *división del problema* global en una serie de subproblemas que deberán asignarse posteriormente a determinados agentes. Es el paso más importante. Por una parte, se debe decidir cómo dividir de forma concreta el problema y de qué manera. Esta tarea puede realizarse por un agente especializado para la tarea, o entre todos los agentes. Después, hay que encontrar a los agentes que puedan resolver cada subproblema. Si éstos se asignan a agentes individuales, no existe ningún inconveniente; pero si hay varios agentes capaces de resolver el mismo problema (esto es, de conocimientos similares), serán necesarias estrategias de cooperación o de decisión de la asignación.

2. La *solución de los subproblemas.* Como cada agente es responsable de la solución del problema asignado, en principio no es necesaria ninguna comunicación ni cooperación entre agentes. Sin embargo, a menudo esto no sucede, puesto que normalmente, el agente no va a ser capaz de resolver el problema de forma autónoma. Los agentes deberán preguntar a otros por problemas que ellos mismos no serán capaces de resolver. A veces, el agente sabrá a qué agente (o conjunto de ellos) debe preguntar; otras veces, el agente no sabrá a quién preguntar, con lo que necesitará del concurso de otros agentes que buscarán por él agentes que posean el conocimiento requerido.

3. La *combinación de subsoluciones* para generar una solución al problema global. Este paso genera problemas parecidos al primer paso; esto es, será necesario un agente capaz de sopesar la importancia de las subsoluciones y mezclarlas para obtener una solución concreta. Esto no siempre tiene por qué suceder; a veces, la mezcla de subsoluciones no generará una solución completa.

Se puede encontrar más información sobre los sistemas multiagente en [**Sycara, 1998**] y [**Barber, 2001**]. Una aplicación jovial de esta tecnología es [**RoboCup web, 2001**].

4.2 Conceptos Básicos

4.2.1 Agentes: definición y propiedades

El término *agente* se ha vuelto muy popular en la industria del software y se utiliza para describir sistemas software que realizan, de forma automática, tareas útiles. Así podemos hablar de *agentes de software* ([**Genesereth and Ketchpel, 1994**]). Un agente es algo que actúa, pero necesitamos diferenciar un agente de un programa ([**Franklin and Graesser, 1996**]). Al ser un área que ha evolucionada rápidamente, se ha dado un sobreuso de éste término debido a la falta de consenso para definirlo ([**Forner, 1993**]). Esto ha propiciado la aparición de numerosos términos sinónimos, como softbot por "software robots" (robots de software), knowbots por "knowledge-based robots" (robots basados en el conocimiento), taskbots por "task-based robots" (robots basados en la tarea), userbots, robots, agentes personales, agentes autónomos y asistentes personales. Pero esta extensa variedad de sinónimos está en parte justificada, ya que los agentes pueden tomar distintos aspectos físicos (robots del mundo real, tanto fijos como móviles, programas en redes de ordenadores, ...) así como juegan distintos papeles o roles (asistentes personales en determinadas tareas, expertos en determinado dominio, ...). También se les califica con múltiples adjetivos ([**King, 1995**]): agentes de búsqueda, agentes de información, agentes de presentación, agentes de navegación, agentes que juegan un papel, agentes de gestión, agentes de búsqueda y recuperación, agentes de diseño y análisis, agentes de prueba, agentes de dominios específicos, agentes de desarrollo y agentes de ayuda.

A pesar de esto, existen algunas propiedades que comúnmente se asocian con el concepto de agente. Estas son :

Autonomía . Los agentes son capaces de operar sin necesidad de intervención directa del usuario. Ellos guían sus actos de acuerdo a sus propias leyes y directivas internas.

Sociabilidad y cooperación . Los agentes se relacionan con otros agentes y con los usuarios. La relación entre el agente y el usuario puede verse como una conversación entre dos, donde el usuario especifica las acciones que quiere realizar y el agente responde y devuelve soluciones. Por otra parte, la comunicación entre agentes se refiere a intercambiar conocimiento (o áreas de conocimiento), posibles soluciones o acciones concretas, de manera que entre todos puedan resolver problemas que un solo agente no puede resolver.

Reacción . Los agentes reciben estímulos del entorno (perciben) y res-

ponden (reaccionan) en un tiempo adecuado a los cambios que ocurran en él. Esto implica modificar reglas en las bases de hechos del agente y en enviar mensajes a otros agentes del entorno.

Proactividad [1]. No simplemente actúan en respuesta al entorno sino que persiguen objetivos. El comportamiento de los agentes está dirigido por una meta, y para su consecución toman la iniciativa en las acciones a llevar acabo, esto es, pueden razonar sobre sus intenciones y planes.

Continuidad temporal . Son procesos que están ejecutándose continuamente.

Adaptabilidad . Los agentes son capaces de adaptarse a los cambios en el entorno.

Personalizables . Los agentes deben educarse para realizar de un modo determinado la tarea que se les ha asignado. En teoría, los agentes deben ser componentes que aprendan y memoricen.

Los agentes además pueden exhibir otras propiedades como *aprendizaje, movilidad, ...*

Si intentamos clasificar los agentes, nos encontramos que existen distintas dimensiones para ello. En los siguientes apartados veremos algunas clasificaciones típicas. Pero una clasificación absoluta es difícil, ya que los agentes se mueven en un espacio multidimensional, y cualquier clasificación será incompleta e inexacta ([**Nwana, 1996**]).

4.2.2 Agentes Deliberativos y Agentes Reactivos

Una primera y simple clasificación de los agentes, según las estrategias de procesamiento, diferenciaría entre dos tipos: agentes reactivos y agentes deliberativos.

Los *agentes reactivos* o *reflexivos* son agentes relativamente simples, en el sentido de que captan (sienten) su mundo y reaccionan (responden reflexivamente) a sus estímulos. Un ejemplo sencillo sería un termostato que reaccionaría a un estímulo (cambio de la temperatura ambiental), pero no tiene la habilidad de realizar planes. Se basan en la idea de aparentar inteligencia sin razonamiento, de forma que los patrones complejos de comportamiento *emergen* de estas interacciones cuando observamos globalmente el conjunto de agentes ([**Brooks, 1991a**] y [**Brooks, 1991c**]).

[1] En el sentido de tomar la iniciativa, de lo opuesto a pasividad: cuando no hay proactividad hay pasividad

Los *agentes deliberativos*, *racionales* o *dirigidos por el objetivo* son más complejos ya que utilizan conocimiento y razonamiento (e incluso aprendizaje) para planificar y conseguir sus objetivos. Suelen tener modelos simbólicos explícitos del entorno y capacidad de razonamiento lógico. A menudo son llamados *agentes BDI*, ("Belief", "Desire" e "Intention"), ya que tienen creencias, deseos e intenciones, junto con complejos estados mentales internos y capacidad de razonamiento ([**Rao and Georgeff, 1995**]). El filósofo Michael Bratman ([**Bratman, 1987**]) distingue esos tres factores (*creencias*, *deseos* e *intenciones*) para determinar los planes, e insiste en que los tres son esenciales. Podemos extender estos conceptos con *objetivos* y *planes*.

Creencias : visión fundamental del agente con respecto a su mundo.

Deseos : opinión del agente de situaciones futuras, derivados directamente de sus creencias.

Objetivos : subconjunto de los deseos del agente en cuyo cumplimiento está actuando.

Intenciones : subconjunto de los objetivos.

Planes : combinación de las intenciones del agente en unidades consistentes.

Para paliar las desventajas y aprovechar las ventajas de los distintos modelos expuestos, aparecen los *agentes híbridos*, que poseen componentes reactivas y deliberativas ([**Maes, 1991a**]). Mientras la componente reactiva se utiliza principalmente para la interacción con el medio, el sistema deliberativo se concentra en la toma de decisiones y la elaboración de planes.

4.2.3 Agentes Estacionarios y Agentes Móviles

También podemos clasificar los agentes por su movilidad, es decir, por su habilidad para moverse por la red. Los *agentes móviles* son procesos (software) que pueden viajar a través de los sistemas haciendo trabajo para sus propietarios. Por contra, los *agentes estacionarios* o *estáticos* no son capaces de abandonar su ubicación original (normalmente el sistema en el que fueron creados); aunque puedan mandar mensajes a objetos remotos, no son capaces de moverse por sí mismos de un ordenador a otro.

Así, estos agentes móviles ([**Green *et al.*, 1997**]) tendrán la capacidad de navegar a través de la red, deteniendo su ejecución para continuar desde otro host. Un agente puede ejecutar cierta tarea en un host, y cuando

necesita algún otro tipo de información, detiene su ejecución, navega a otro host y sigue ejecutando la tarea asignada. Estas tareas serán ejecutadas en modo local, ya que el agente se encontrará en la máquina remota. Estos tipos de agentes dependen de sitios que permitan su ejecución. Tienen la ventaja de que una vez lanzados a la red, pueden navegar por sí mismos (saben a dónde ir) y ejecutar acciones cuando estimen convenientes. Diremos que tienen la capacidad de *migrar*. En este sentido, la librería que quizá ha significado un relativo éxito y permite implementar de manera eficiente agentes móviles, es la de la sección japonesa de IBM que te permite programar *Aglets* ([**Aglets web, 2001**]), que son applets capaces de navegar por la red. Los aglets son objetos Java que pueden moverse de un host en internet a otro. Cuando un aglet se mueve se lleva tanto su código como sus datos (información de su estado).

4.2.4 Solución de Problemas

Los agentes inteligentes deben actuar de manera que el entorno experimente una serie de estados tales que permitan obtener un máximo en la medida de rendimiento. La formulación de metas, tomando como base la situación de un momento dado, es el primer paso en la solución de problemas. Además, el agente quizás quiera tomar una determinación en relación con otros factores que afectan lo deseable de las diversas maneras de alcanzar una meta. Se consideran que las metas son conjuntos de estado del mundo: sólo aquellos que permiten el logro de la meta. Se puede considerar que las acciones son las causantes de la transición entre uno y otro estado del mundo; por ello, el agente tiene que determinar qué acciones permiten obtener el estado correspondiente a una meta. Pero, para ello, primero tiene que decidir qué tipo de acciones y estados habrá que tomar en consideración. La formulación de un problema, por tanto, es el proceso que consiste en decidir qué acciones y estados habrán de considerarse, y es el paso que sigue a la formulación de metas. En términos generales, cuando un agente tiene ante sí diversas opciones inmediatas cuyo valor ignora, para decidir lo que debe hacer primero tiene que evaluar las diversas secuencias de acciones posibles que le conducen a estados cuyo valor se conoce, y luego decidirse por la mejor. El diseño del agente, al final, se reduce simplemente a formular, buscar y ejecutar. Después de formular una meta y el problema que hay que resolver, el agente solicita un procedimiento de búsqueda que le permita resolverlo.

Básicamente, existen cuatro tipos de problemas:

Problemas de un solo estado . El agente tiene información suficiente como para determinar con exactitud el estado en el que se encuentra (el mundo es accesible); se supone que conoce de manera com-

pleta el resultado producido por cada una de sus acciones. Por lo tanto, podrá calcular exactamente en qué estado se encontrará después de una determinada secuencia de acciones.

Problemas de estado múltiple . El agente sabe cuál es el resultado de cada una de sus acciones, pero el acceso al mundo es limitado. En este caso, el agente deberá razonar en términos de los conjuntos de estados a los que puede llegar, en vez de pensar en función de estados únicos.

Problemas de contingencia . Hay veces que la ignorancia impide al agente encontrar una secuencia de solución garantizada. El agente ahora debe calcular un árbol de acciones, en vez de una sola secuencia de ellas. Por lo general, cada una de las ramas del árbol se refiere a una posible contingencia que pudiera surgir. Muchos de los problemas del mundo real, físico, son problemas de contingencia, puesto que es imposible hacer predicciones exactas.

Problemas de exploración . El agente no cuenta con información sobre los efectos de sus acciones. Tendrá que experimentar, descubrir poco a poco qué tipo de acciones emprender y qué tipo de estados existen. El anterior es un tipo de búsqueda, pero en el mundo real no es un modelo. Ejecutar un paso en el mundo real, en vez de hacerlo en un modelo, entraña grandes peligros para un agente ignorante. Si logra sobrevivir, el agente aprenderá un mapa del ambiente, mapa que puede utilizar para resolver otros problemas que se le presenten.

Los problemas de un solo estado y los de estado múltiple se resuelven con las mismas técnicas de búsqueda. Para los problemas de contingencia es necesario emplear algoritmos más complejos.

4.3 Arquitecturas de Sistemas Multiagente

Si tenemos sistemas con agentes que interactúan, estos agentes deben poder comunicarse, pero la forma en que se llevará a cabo esa comunicación depende en gran medida de la manera de organizarlos. Cuando un agente necesita comunicarse con otro puede dirigirse directamente a él, pero, para ello necesita hablar un lenguaje común, o puede utilizar un intérprete, que le entienda a él y al agente con el que quiere comunicarse. Estas son, fundamentalmente, las dos formas de organización de agentes que se están utilizando: comunicación directa y comunicación asistida. Pasemos a describirlas brevemente.

4.3.1 Comunicación Directa

Una *organización de agentes por comunicación directa* se caracteriza por agentes que manejan su propia coordinación, la cual tiene la ventaja de que no depende de otros programas y la desventaja de que aumenta el grado de complejidad en la implementación de cada agente. Dentro de esta forma de organización existen dos arquitecturas:

- En la arquitectura de *red de contratos*, los agentes cuando necesitan algún servicio distribuyen *solicitudes* a diferentes agentes. Los receptores de estas solicitudes las evalúan y lanzan *ofertas*, las cuales son usadas por los agentes solicitantes para decidir con qué agente realizar un *contrato* (ver página 74). Esta arquitectura es costosa por la cantidad de mensajes que se requieren enviar.

- En la arquitectura de *especificación compartida*, los agentes no hacen solicitudes de servicio sino que proveen de información a otros agentes acerca de sus *capacidades* y *necesidades*; y cuando surge la necesidad de un servicio, esta información es utilizada por los agentes para coordinar sus actividades. El número de mensajes que se intercambian se reduce considerablemente en comparación con la arquitectura anterior.

Una desventaja de la comunicación directa es el coste. Si el número de agentes es pequeño no hay problema. Pero cuando éste crece (como en el caso de la utilización de agentes en Internet) el coste del procesamiento de todos los mensajes (ofertas) es prohibitivo. Una posible solución sería organizar los agentes de alguna manera. La técnica más simple de organización es en forma de arquitectura cliente/servidor, de forma que agrupamos una serie de agentes con un agente que tiene una visión general del sistema. Otra desventaja es la complejidad de implementación, ya que cada agente es responsable de su propia negociación.

4.3.2 Coordinación Asistida

Otra forma de organizar a los agentes es mediante la *coordinación asistida*. Un ejemplo bastante popular de este tipo de organización es la de arquitectura conocida como *sistema federado*, en donde en lugar de que los agentes se comuniquen directamente, lo hacen por medio de *intérpretes* o *facilitadores*[2], y éstos se comunican entre ellos (ver figura 4.1). En

[2]Aunque el término intérprete es más intuitivo, utilizaremos la traducción literal del término utilizado en inglés "facilitator".

esta arquitectura, los agentes generalmente utilizan ACL ("Agent Communication Language") (ver apartado 4.4.2) para comunicar sus necesidades y habilidades a su facilitador local, quien se encarga de encontrar la ruta correcta por la cual hace llegar solicitudes a otros facilitadores, quienes a su vez pasan las solicitudes a alguno de los agentes de su dominio que pueda satisfacer la solicitud. Una característica importante de la arquitectura de federación es que soporta la interacción anónima entre los agentes a cambio de que éstos cedan algo de su autonomía al facilitador y de que se adhieran a condiciones adicionales. Para cada agente debe parecer que existe sólo un agente que maneja todas sus solicitudes directamente.

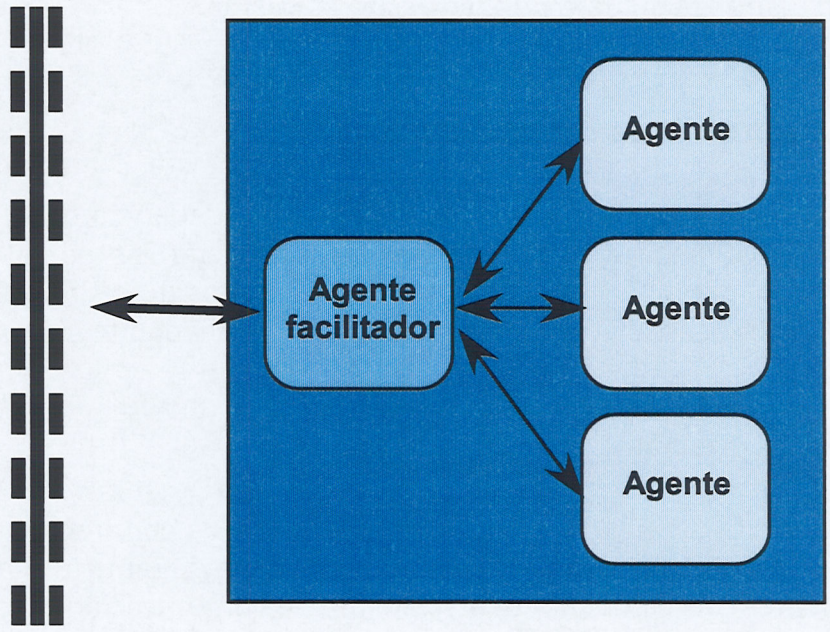


Figura 4.1: Arquitectura de Coordinación Asistida por Facilitadores

Los servicios que el facilitador provee son los siguientes:

- Encontrar la identidad de agentes por su nombre.
- Encontrar la identidad de los agentes capaces de desempeñar una tarea.
- Enviar un mensaje a un agente específico.
- Hacer uso de las especificaciones para manejar las solicitudes y dar la impresión de que él provee todos los servicios.
- Descomponer solicitudes complejas en subproblemas, obtener las respuestas de cada supbroblema y combinarlas para obtener la respuesta a la solicitud original.

- Traducir del vocabulario de un agente al vocabulario de otro.
- Monitorizar su base de conocimiento para determinar si una solicitud puede ser satisfecha.

4.4 Comunicación entre Agentes

Como ya hemos dicho la comunicación es la clave de los sistemas multiagente ([**Haddadi, 1996**]). Vamos a tratar tanto los métodos de comunicación como el lenguaje utilizado para la misma.

4.4.1 Métodos de Comunicación

Existen diversos métodos para que los agentes puedan comunicarse. El más simple consistiría en hacer llamadas a procedimientos del agente. Sin embargo, debido a las características implícitas de los agentes, este método es desaconsejable para un sistema medianamente complejo.

Sistemas de pizarra

La arquitectura de pizarra es ampliamente utilizada en Inteligencia Artificial. En un sistema con múltiples fuentes de conocimiento (o agentes independientes) necesitamos un mecanismo de comunicación. La *pizarra* ("blackboard") es una estructura de datos que es usada como mecanismo general de comunicación entre las múltiples fuentes de conocimiento y es gestionada y arbitrada por un controlador ([**Hayes-Roth, 1985**], [**Nii, 1986b**], [**Nii, 1986a**] y [**Jagannathan *et al.*, 1989**]). Así, será un área de trabajo común a los agentes donde poder intercambiar información, datos y conocimiento. Al igual que las pizarras de nuestras aulas, son dispositivos para compartir información, con múltiples escritores y múltiples lectores.

Como cada agente trabaja con su parte del problema, acudirá a la pizarra para ver nueva información puesta por otros agentes y, a su vez, pondrá sus resultados. La forma de funcionar será: un agente inicia una comunicación escribiendo algún tipo de información en la pizarra; en ese momento, estos datos están disponibles para todos los agentes del sistema; cada agente accederá a la pizarra eventualmente para comprobar si hay información nueva; normalmente, cada agente no recogerá toda la información que se vaya escribiendo en la pizarra, sino que sólo obtendrá aquélla que le interese, tal vez por pertenecer a conocimiento afín.

Como se puede observar, no hay comunicación directa entre agentes; por esto mismo, cada agente se ve obligado a resolver de forma autónoma su subproblema, bajo su responsabilidad. En las pizarras no existen áreas privadas; los agentes pueden escribir la información que deseen, y pueden acceder a toda la información contenida en la pizarra. Por tanto, si hay un número grande de agentes, la información contenida en la pizarra crece exponencialmente; y por otra parte, los agentes deberán buscar en una gran cantidad de información por cada acceso que realicen a la pizarra. Para optimizar este proceso, existen métodos más complejos (ampliaciones del original) para definir regiones en la pizarra, de manera que un agente sólo ve la región de la pizarra que tenga asignada. Un sistema multiagente puede disponer de varias pizarras.

Aunque no hay una definición específica de las estructuras de datos que debe contener la pizarra, es obvio que por lo menos se debe asegurar que el resto de agentes deben ser capaces de leer y entender los datos incluidos en el sistema. Además, se deberán incluir informaciones adicionales que faciliten el proceso de solución del problema global.

Otro dato a tener en cuenta es la calidad de las contribuciones individuales de los agentes al sistema. Como cualquier agente tiene potestad para escribir cualquier cosa en la pizarra y esto podría generar problemas, estudios posteriores de la idea original han incluido una cierta coordinación para el mantenimiento de la pizarra. Para ello, se ha introducido la idea del *moderador*, que se encarga de supervisar el estado del problema, los siguientes subproblemas a ser resueltos y además trata de organizar el trabajo entre y hacia los agentes. Cualquier agente puede usar la pizarra para leer los subproblemas generados. Si está interesado en resolver alguno, lo indica usando un registro de activación de fuente de conocimiento (KSAR, "Knowledge Source Activation Record") en una base de datos. El moderador controla y evalúa el conocimiento de la base de conocimiento para seleccionar a los agentes más capacitados para resolver el subproblema. Otra mejora es incluir un *dispatcher* que se encarga de informar a los agentes interesados de nueva información que se escribe en la pizarra, en vez de que éstos accedan regularmente a ella.

Los sistemas de pizarra ofrecen un sistema flexible para la cooperación y comunicación entre las unidades que componen un sistema distribuido, ya que son independientes de las estrategias de cooperación. Sin embargo, la estructura central del sistema es un obstáculo para sistemas orientados a redes, puesto que la sola lectura de la información contenida en la pizarra representa un cuello de botella importante en la red.

Sistemas basados en paso de mensajes

En este caso, los agentes intercambian mensajes para establecer la comunicación y cooperación usando protocolos definidos. En principio, el sistema actuaría de forma similar a una arquitectura cliente-servidor, donde un agente emisor envía un mensaje a otro agente receptor. No existe, de esta forma, ningún receptáculo de información, de manera que ningún otro agente puede leer dicho mensaje.

Para definir el sistema, es necesario clarificar dos puntos importantes. Por una parte, es necesario definir un protocolo de comunicación de mensajes. Este punto no representa ningún avance importante. Lo realmente importante para el intercambio de conocimiento consiste en integrar una semántica en el lenguaje de comunicación. Para ello, se crean lenguajes basados en la teoría de *actos de habla* ("speech acts") ([**Searle, 1969**]), con la que se construye una capa lingüística y se formalizan las acciones lingüísticas de los agentes. La teoría de actos de habla ha contribuido en gran medida al entendimiento de la relación entre el estado interno de un agente y las expresiones que intercambia con otros agentes. La teoría se basa en la observación de que las oraciones expresadas por humanos durante la comunicación no siempre aseveran un hecho, sino que en realidad tratan de transmitir una creencia o conocimiento, una intención o un deseo.

4.4.2 Lenguajes de Comunicación entre Agentes

Una de las propiedades de los agentes (página 57) es la sociabilidad y cooperación, lo que implica que los agentes deben comunicarse con otros agentes, incluso con los humanos, por medio de un lenguaje. El uso de un lenguaje de comunicación común facilita la creación de software interoperable porque permite disociar la implementación de la interfaz; esto es, siempre que un grupo de programas se rija por un lenguaje de comunicación estándar, éstos pueden interoperar independientemente del lenguaje en el que fueron implementados.

Para diseñar estos lenguajes existen dos métodos a seguir. El método procedural se basa en la idea de que la comunicación se puede modelar más adecuadamente como un intercambio de directivas. De esta manera se pueden transmitir no sólo comandos sino también programas enteros. Usualmente su ejecución es eficiente y directa. Las desventajas principales con este método son el requerimiento de información sobre el receptor del mensaje, la cual no siempre está disponible, y el carácter unidireccional del procedimiento, cuando la mayoría de las veces es conveniente que los agentes compartan información. Por el contrario, el

método declarativo establece que la comunicación se modela de forma más adecuada utilizando enunciados declarativos. Este método para ser útil requiere que el lenguaje sea lo suficientemente expresivo como para comunicar diferentes clases de información, incluyendo procedimientos, que sea compacto y que no se requiera que el vocabulario crezca demasiado para seguir manteniendo la comunicación. Un ejemplo de este tipo de lenguaje es ACL, desarrollado por ARPA.

ACL ("Agent Communication Language")

Como resultado del esfuerzo por crear un lenguaje que permitiera la interoperación entre agentes autónomos distribuidos surgió el lenguaje llamado ACL ("Agent Communication Language") desarrollado por investigadores el grupo "Knowledge Sharing Effort" ([**KSE web, 2001**]). ACL tiene tres componentes:

1. Un vocabulario.
2. Un lenguaje de contenido (KIF, "Knowledge Interchange Format").
3. Un lenguaje de comunicación (KQML, "Knowledge Query Manipulation Language").

Un mensaje de ACL es un mensaje en KQML que consta de una directiva de comunicación y un contenido semántico en KIF construido con los términos del vocabulario.

Vocabulario

El vocabulario de ACL es un diccionario de palabras apropiado para áreas de aplicación comunes ([**Gruber, 1991**]). Cada palabra en el diccionario tiene una descripción (escrita en lenguaje natural) que es usada por las personas para entender su significado y una anotación formal (escrita en KIF) que es usada por los programas. El diccionario es abierto, es decir, es posible añadir nuevas palabras dentro de áreas existentes y en nuevas áreas de aplicación. La existencia de este diccionario no significa que solamente hay una manera de describir un área de aplicación. Un diccionario puede contener múltiples ontologías para un área dada y un agente puede utilizar la ontología que le sea más conveniente. Las definiciones formales asociadas con cualquiera de estas ontologías pueden ser utilizadas por los agentes para traducir mensajes que usan una

ontología en específico a mensajes que usan otras ontologías. Cuando se comparte información en una comunidad se debe tener un acuerdo sobre el significado de los símbolos. Si esta comunidad es pequeña es posible que todos los objetos que se comunican utilicen un vocabulario único. Sin embargo, si la comunidad es extensa, es muy posible que sea necesario soportar una colección de vocabularios.

KIF ("Knowledge Interchange Format")

KIF, como se ha visto en el apartado 1.7 (página 16) es una versión en notación prefija del cálculo de predicados de primer orden con varias extensiones para incrementar su expresividad.

KQML ("Knowledge Query and Manipulation Language")

Aunque es posible diseñar un marco de trabajo completo para la comunicación en el que todos los mensajes tengan la forma de oraciones en KIF, esto sería ineficiente, ya que se requeriría incluir información implícita sobre el agente que envía el mensaje y sobre el que lo recibe, debido a que la semántica de KIF es independiente del contexto. La comunicación se hace más eficiente si se provee de una capa lingüística en la que el contexto se tenga en cuenta. Esta es la función de KQML ([**KQML web, 2001**], [**Finin *et al.*, 1994**] y [**Labrou and Finin, 1997**]). KQML es un lenguaje basado en la teoría de actos de habla. Fue concebido como un formato de mensajes y como un protocolo que maneja los mensajes para permitir a un programa identificar, conectarse e intercambiar información con otros programas. En términos lingüísticos se puede decir que KQML se enfoca principalmente a la parte pragmática de la comunicación. Puede ser usado como un lenguaje para programas de aplicación que interactúen con un sistema inteligente o para dos o más sistemas inteligentes que compartan conocimiento en la resolución cooperativa de problemas. Y, de momento, es el modelo para comunicaciones entre agentes más usado.

Tres características importantes de KQML son:

1. Los mensajes de KQML son opacos al contenido de lo que transportan, esto es, los mensajes en KQML no comunican únicamente oraciones en un lenguaje, sino que comunican una *actitud* acerca del contenido (por ejemplo, afirmación, solicitud, pregunta).

2. Las primitivas del lenguaje se llaman *performativas*[3], las cuales indican las acciones u operaciones permitidas que los agentes pueden

[3]Término recogido de la teoría de actos de habla. Aunque los términos directiva o directriz son más comprensibles, utilizaremos una mala traducción literal del término utilizado en inglés "performative".

utilizar cuando se comunican.

3. El entorno en el que los agentes se comunican con KQML puede ser enriquecido con un tipo de agentes especiales llamados *facilitadores* (descritos en los sistemas federados, ver la figura 4.1 en la página 63), que son una clase especial de agentes que coordinan las interacciones entre los otros agentes.

KQML es un lenguaje que se divide en tres capas o niveles en los cuales codifica la información (figura 4.2):

1. La **capa de contenido** se relaciona con el contenido real del mensaje escrito en el lenguaje de representación propio de cada agente. Un mensaje en KQML puede tener cualquier lenguaje de representación incluyendo lenguajes expresados como cadenas en ASCII y aquellos expresados utilizando una notación binaria. Cualquier implementación de KQML ignora la parte del contenido del mensaje excepto para determinar dónde termina.

2. La **capa de comunicación** codifica un conjunto de características del mensaje, las cuales describen los parámetros de la comunicación de bajo nivel, tales como la identidad del agente que envía el mensaje y la del que lo recibe, así como un identificador único asociado con la comunicación.

3. La **capa de mensaje** se utiliza para codificar el mensaje que una aplicación desea transmitir a otra. Esta capa forma el corazón del lenguaje y determina las clases de interacciones que se pueden tener con un agente que hable KQML. La función principal de la capa de mensaje es identificar el protocolo que se va a usar para entregar el mensaje (síncrono o asíncrono) y proporcionar una performativa que el transmisor le agrega al contenido. Además de esto, ya que el contenido es opaco a KQML, en esta capa también se incluyen características opcionales que describen el lenguaje del contenido, la ontología que se está asumiendo y alguna clase de descripción del contenido. Estas características hacen posible que las implementaciones de KQML analicen, redirigan y entreguen el mensaje apropiadamente aún cuando su contenido sea inaccesible.

Todo mensaje en KQML se inicia con una performativa que indica el tipo de comunicación y un conjunto de argumentos opcionales, llamados *parámetros*, entre los que se encuentra el contenido real del mensaje así como otros que describen el propio contenido. Existe un conjunto de performativas estándar con un significado al que toda implementación de KQML debe adherirse (tablas B.2 y B.3, página 167). Estas performativas estándar se dividen en tres grupos:

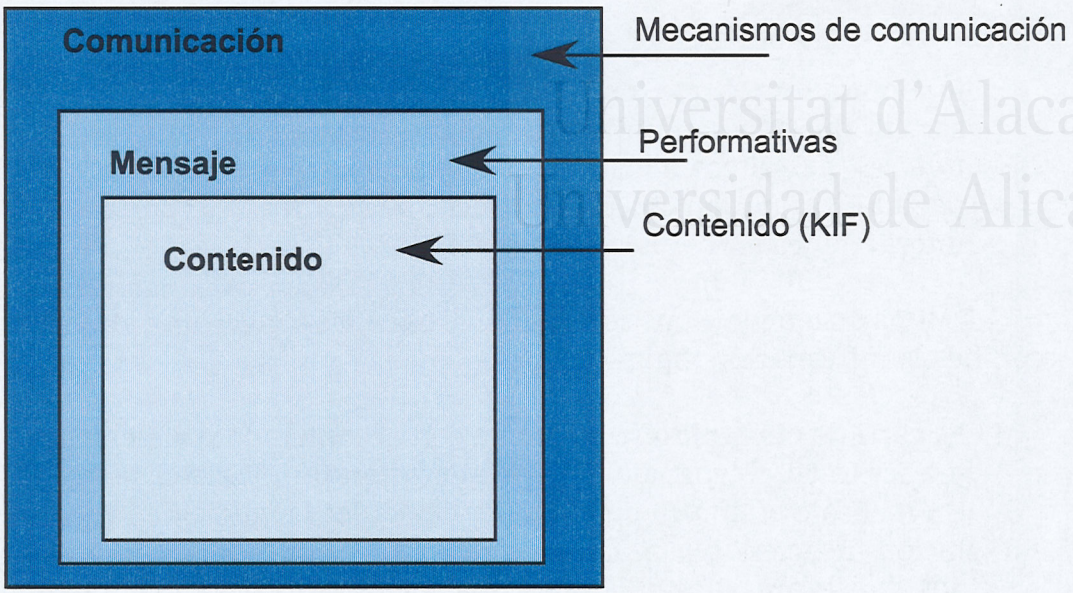


Figura 4.2: Capas del Lenguaje KQML

1. Las performativas de **discurso** (`ask-if`, `ask-all`, `ask-one`, `tell`, `deny`, `achieve`, `advertise`, `subscribe`, entre otras), empleadas en el contexto de un intercambio de información y conocimiento entre dos agentes.

2. Las performativas de **intervención** y **mecánica de la conversación** (`error`, `sorry`, `ready`, `next`, `discard`, `rest`, `standby`), cuyo papel es intervenir en el curso normal de una conversación.

3. Las performativas de **red** y de **facilitación** (`register`, `forward`, `broadcast`, `recommend-one`, `recruit-all`, entre otras), los cuales, estrictamente hablando no son actos del habla, pero permiten a los agentes encontrar otros agentes capaces de procesar sus mensajes.

Los nombres de los parámetros empiezan con ":" y deben estar seguidos por el correspondiente valor del parámetro. Los posibles parámetros con su correspondiente descripción se pueden ver en la tabla B.1 de la página 166.

El siguiente es un ejemplo de un mensaje KQML donde un agente que se identifica como `jorge` solicita a un `facilitador` que envíe un mensaje proveniente del mismo `jorge` y escrito en KQML a otro agente `ana`, asumiendo la ontología `ontologia-fdl`. A su vez el mensaje que `jorge` quiere hacer llegar a `ana` a través de `facilitador` indica que `jorge` desea que `ana` haga cierto en su entorno lo que está en el contenido del mensaje (`estado=suspendido`) y que le responda con un mensaje identificado

por `id1`. El parámetro `content` define el nivel de contenido; los parámetros `from`, `to`, `sender` y `receiver` especifican información del nivel de comunicación; y `language` y `ontology` forman parte del nivel de mensaje.

```
(forward
   :from jorge
   :to ana
   :sender jorge
   :receiver facilitador
   :language kqml
   :ontology ontologia-fdl
   :content (achieve
        :sender jorge
        :receiver ana
        :reply-with id1
        :content (estado=suspendido)
   )
)
```

Intuitivamente, podemos decir que cada mensaje en KQML es una pieza de diálogo entre emisor y receptor, y KQML proporciona el soporte para una amplia variedad de tipos de diálogos. El conjunto de performativas es extensible y no todas son necesarias. Un agente puede elegir manejar sólo unas pocas performativas de las descritas, e incluso implementar algunas adicionales. Sin embargo, cualquier implementación de las performativas reservadas debe ser utilizada de la manera descrita.

Para dotar a la comunicación de infraestructuras estándares de más bajo nivel para lograr la coordinación y la interpoerabilidad real entre aplicaciones en forma más global y robusta, existen desarrolladores que aconsejan el uso de CORBA ("Common Object Request Broker Architecture") por debajo del KQML.

4.5 Cooperación y Competición entre Agentes

En un sistema multiagente, los agentes van a tener que ser capaces de cooperar entre ellos, normalmente para obtener soluciones de problemas que el propio agente no es capaz de encontrar (CMAS "Cooperative Multi-Agent Systems"). Por otro lado, desde el momento en que nosotros tenemos agentes que cumplen nuestras órdenes, otras personas tendrán agentes que cumplirán las suyas. Y mientras nuestros agentes quieren lo mejor para nosotros, los de los demás también intentarán lo mejor para sus propietarios (SMAS "Self-Interested Multi-Agent Systems"). En este punto se encuentran principalmente estrategias de negociación que permiten resolver conflictos y asignar tareas ([**Smith and Davis, 1981**] y

[**Rosenschein and Zlotkin, 1994**]).

4.5.1 Negociación

La negociación es importante en la modelización de sistemas multiagente. Podemos considerarla como un proceso de comunicación entre un grupo de agentes para cumplir un contrato mutuamente aceptado ([**Bussmann and Muller, 1992**]). La negociación en los sistemas multiagente se observa desde varios puntos de vista. Por una parte, tanto la asignación de subproblemas como la asignación de recursos puede contemplarse como una negociación; por otro lado, también deben existir negociaciones entre agentes individuales para desarrollar una cooperación en beneficio del sistema. La negociación puede ser *competitiva* o *cooperativa*, según el comportamiento de los agentes individuales (CMAS "Cooperative Multi-Agent Systems" y SMAS "Self-Interested Multi-Agent Systems").

Existen varias situaciones diferentes a la hora de entablar una negociación:

- Ninguno de los agentes obtiene beneficio a través de la negociación, puesto que cada agente sigue su propio objetivo y no existen dependencias directas entre ambos objetivos.
- Al menos uno de los agentes logra el objetivo de forma más rápida o con menos esfuerzo.
- Existen situaciones reales de conflicto entre los agentes, debido a recursos, logros de objetivos similares, etc.

Una clasificación interesante desde el punto de vista individual del agente divide en cuatro las posibles formas de actuar en la negociación:

Cooperación simétrica La negociación produce resultados mejores que los que cada agente obtendría de manera individual.

Compromiso simétrico Los agentes logran el objetivo independientemente y negocian un compromiso entre ambas partes, de manera que se degrada el resultado en comparación con las soluciones independientes. Es esencial un compromiso debido a que no se puede obviar a ninguno de los agentes.

Cooperación/compromiso no simétrico Uno de los agentes obtiene una mejora de su resultado, mientras que el otro empeora su solución.

Conflicto No se puede llegar a un acuerdo razonable debido a que los objetivos entran en conflicto. La negociación debe terminar sin obtener un resultado claro.

En las estrategias de negociación, actualmente están tomando relevancia los algoritmos genéticos, puesto que cada agente podría generar un número aleatorio de estrategias de negociación y aplicarlas, utilizando posteriormente aquellas que den mejor resultado. Sin embargo, el sistema podría requerir de un número grande de estrategias de negociación, con lo que puede resultar inviable. También está creciendo el uso de la teoría de juegos (apartado 6.3, página 110) en el ámbito de la negociación, asumiendo negociaciones más bien agresivas. Los conceptos claves de teoría de juegos aplicada a la negociación son:

Función de Utilidad la *utilidad* puede ser definida como la diferencia entre el *beneficio* de conseguir un objetivo y el *coste* invertido para lograrlo ([**Haddawy and Hanks, 1992**] y [**Bacchus and Grove, 1996**]).

Espacio de Transacciones una *transacción* es una acción que un agente hace y que lleva una utilidad asociada.

Estrategias y Protocolos de Negociación un protocolo de negociación define las reglas que gobiernan la negociación, incluyendo cómo y cuando finaliza la misma.

4.5.2 Planificación Global Parcial

La *planificación global parcial* ("Partial Global Planning") ([**Durfee, 1988**]) ha sido una de las primeras estrategias utilizadas en agentes para coordinar los planes locales mientras trabajan para conseguir objetivos globales. Es una metodología que asume que un nodo conoce el estado actual y el objetivo del resto de los nodos. De esta manera, cada nodo sabe el grado de conocimiento del resto y puede generar sus propias conclusiones respecto a su trabajo; el nodo sabe a quién dirigirse en caso de no ser capaz de resolver subproblemas, y sabe que el otro nodo lo puede resolver. Tiene un plan local de otro nodo que le indica que lo puede resolver. La principal ventaja del sistema es el comportamiento dinámico del mismo, ya que todos los planes pueden adaptarse a nuevos cambios de entorno. Sin embargo, también obliga a que estos cambios sean notificados

a todos los nodos del sistema, ya que los cambios afectan también a su trabajo. Además, el sistema evita redundancias, puesto que si un nodo realiza una tarea similar a la de otros, estas tareas pueden reorganizarse.

4.5.3 Sistema de Red de Contratos

Ya hemos visto en la página 62 una visión muy simplificada de la aproximación de red de contratos. Una *red de contratos* ("Contract Net") ([**Smith, 1980**] y [**Davis and Smith, 1983**]) se compone de múltiples nodos formados por los agentes que forman el sistema. El modo de trabajo es similar a un mercado, donde un nodo presenta una oferta (propuesta) a los nodos interesados en realizar un trabajo. La finalidad es aprovechar los recursos ofreciendo la tarea al mejor nodo disponible.

Los nodos del sistema se componen de varias partes:

- Una *base de datos local* que contiene el conocimiento base del nodo e información sobre el estado actual de las negociaciones y del estado de resolución del problema. El resto de componentes utiliza la base de conocimiento para realizar sus propias tareas.
- El *procesador de comunicaciones* se encarga de recibir y enviar los mensajes al resto de los nodos. Es el único componente que tiene una conexión directa con la red.
- El *procesador de contratos* controla las tareas apropiadas para poder realizar ofertas, envía las aplicaciones y finaliza los contratos.
- El *procesador de tareas* es responsable de la ejecución y resolución de la tarea asignada al nodo; obtiene el problema a resolver del procesador de contratos, utiliza la base de datos para determinar una solución y se la pasa al procesador de contratos.

Los pasos básicos del protocolo de red de contratos son:

1. Un agente administrador realiza una oferta pública sobre un problema que se deberá resolver. Para ello, lanza un mensaje en un determinado protocolo.
2. La oferta es evaluada por los procesadores de contratos de los nodos del sistema. Este evalúa las capacidades del nodo y responde en caso de poder resolverlo.

3. El administrador deberá elegir, de entre las ofertas que le hayan llegado, la mejor para cerrar el trato. Envíará un contrato al nodo elegido y se encargará de notificar a todos el resultado de la negociación.

4. El agente contratado realizará la tarea y devolverá los resultados.

Existen ampliaciones del modelo original, sobre todo en lo referente a la oferta pública, ya que puede implicar un cuello de botella amplio al tener que estudiar múltiples ofertas, etc. Existen otros protocolos de negociación ([**Kumar and Feldman, 1998**]): *subastas* ("auctions"), *pactos* ("bargaining"),...

4.5.4 El Matchmaker y el Broker

Hay dos especializaciones de agentes que permiten dar mayor flexibilidad al sistema. Si un agente no sabe exactamente dónde buscar la solución a un subproblema, envía una notificación al *matchmaker* o facilitador ([**Kuokka and Harada, 1995**]) que se encargará de buscar un agente que sea capaz de resolver el problema del primer agente. Cuando el matchmaker encuentra al agente adecuado, envía su nombre al agente que quería resolver el problema, que ya se pondrá de acuerdo con aquél y negociará posibles soluciones. Hay otras ocasiones, sin embargo, donde un agente requiere a otro para que le dé la solución directamente. Esta es la labor del *broker*, que recibe la solicitud del agente que quiere resolver el subproblema, busca a algún agente capaz de resolverlo, negocia con él la posible solución y se la envía al agente original.

Volver al índice/Tornar a l'índex

Figura 4.3: Esquema de Sistemas de Agentes

Capítulo 5

Modelo de Integración de Agentes Inteligentes

> Imagine que se han descubierto un conjunto de reglas aplicables a todos los participantes, llamadas "óptimas" o "racionales". Estas reglas son desde luego óptimas, en el caso de que los demás participantes se atengan a ellas. Queda por ver lo que sucedería si uno de los participantes no siguiera las reglas. Si de aquí resultan consecuencias beneficiosas para ellos, y en particular perjudiciales para los que cumplen las reglas, se pondría muy en duda la "solución" mencionada anteriormente ... Independientemente de la manera en que formulemos los principios básicos y la justificación objetiva de un "comportamiento racional", habrá que tener en cuenta las condiciones que conllevan todos los comportamientos posibles de "los otros".
>
> *Theory of Games and Economic Behavior.*
> John von Neumann y Oskar Morgenstern

Esta parte describe el *Sistema de Agentes Inteligentes en Java*, software desarrollado para integrar todos los aspectos tratados en los capítulos anteriores. En un primer momento describiremos el modelo y justificaremos su configuración. Posteriormente pasamos a comentar las herramientas que dan soporte al mismo. Finalizando el capítulo con los aspectos técnicos del Sistema de Agentes Inteligentes propuesto y desarrollado en Java.

5.1 Modelo Propuesto

Una vez caraterizados en los capítulos anteriores los aspectos teóricos que nos interesan, vamos a decidir qué características son las que vamos a incorporar a nuestro sistema. En la figura 5.1 podemos ver un esquema del modelo de sistema de agentes que proponemos y que pasaremos posteriormente a implementar.

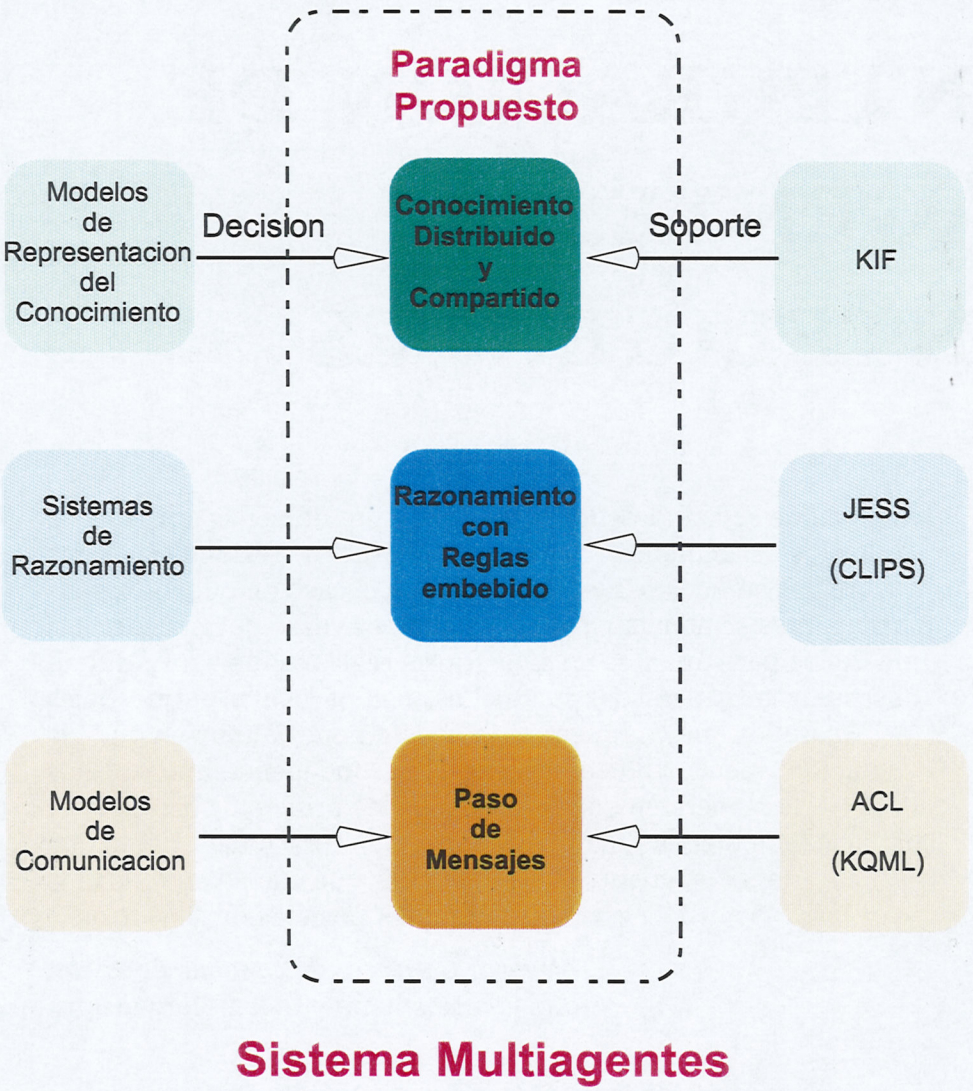


Figura 5.1: Esquema del Modelo de Agentes Propuesto

Vemos que por cada una de las tres grandes áreas que hemos tratado tenemos que decidirnos por un modelo concreto:

- Así, como modelo de representación del conocimiento nos decantamos por un modelo simbólico, que combina la potencia expresiva

de la lógica con los beneficios de la representación de objetos. Al mismo tiempo queremos que el conocimiento esté distribuido por el sistema y sea compartido por los distintos agentes. El soporte práctico sobre el que desarrollaremos nuestro modelo es el lenguaje KIF ("Knowledge Interchange Format").

- En cuanto al sistema de razonamiento, utilizamos un sistema basado en reglas, que es sencillo de entender y se puede empezar a trabajar con él sin esfuerso, pero al mismo tiempo es suficientemente potente y flexible para ir incorporando las mejoras que consideremos oportunas (incertidumbre, información cambiante, manejo de creencias, ...). El soporte elegido ha sido el lenguaje declarativo CLIPS, o mejor dicho, el módulo JESS ("Java Expert System Shell") que permite integrar ficheros CLIPS en programas Java. Por ello hemos implementado unos agentes especializados genéricos, que pueden cargar un fichero con conocimiento y representar distintos roles.

- Finalmente, en cuanto a las características de los agentes, hemos optado por unos agentes deliberativos estacionarios que se comunican por paso de mensajes. Al poder asumir distintos roles los agentes, podemos utilizar fácilmente algunos de ellos como facilitadores. El soporte para la comunicación entre los agentes (ACL) nos la da el lenguaje KQML ("Knowledge Query and Manipulation Language").

5.2 Herramientas

5.2.1 Marco Genérico

Java

En estos momentos **Java** es un lenguaje ampliamente extendido, y una elección acertada a la hora de construir sistemas de agentes ([**Bigus and Bigus, 2001**]), ya que sus características lo hacen apropiado para implementar agentes o sistemas multiagente. Java ([**Deitel and Deitel, 1998**], [**Morgan, 1999**], [**Bishop, 1999**] y [**Weiss, 2000**]) es un lenguaje orientado a objetos, creado por Sun Microsystems ([**Sun Microsystems, 2001**]) y diseñado originariamente para programar en tiempo real electrodomésticos de consumo. Esto obligaba a un código independiente de la máquina y de tamaño reducido. Pero su gran popularidad se debe a su uso en Internet, al incorporarse en los navegadores.

Java se ejecuta sobre una máquina virtual denominada *Java Virtual Machine* (JVM) que interpreta el código neutro convirtiéndolo al código particular de la CPU, de ahí su lema "Write Once, Run Everywhere".

Java es un verdadero lenguaje orientado a objetos, sin funciones, ni procedimientos ni variables globales, y sin punteros. Todo (o casi todo) en Java es un objeto, lo que contribuye a su robustez. Además podemos organizar las clases que encajan lógicamente y que pueden interaccionar entre ellas en *paquetes*.

Veamos algunas características de los agentes, que son fácilmente realizables en Java:

Autonomía Las aplicaciones Java son procesos separados y autónomos que se pueden comunicar entre sí.

Inteligencia Representaciones de conocimiento (ver apartado 1.1) del tipo redes semánticas, marcos y reglas de producción son implementadas en Java de forma fácil y natural.

Movilidad La portabilidad del código Java permite que éste sea enviado y ejecutado remotamente.

JATLite y JAFMAS

Existen dos paquetes implementados en Java que ofrecen facilidades para crear sistemas de agentes.

JATLite ("Java Agent Template, Lite", **[JATLite web, 1998]**) es un paquete de programas escritos en Java que permiten al usuario crear nuevo software de agentes que se comunican robustamente a través de Internet. Provee el AMR ("Agent Message Router"), que es una aplicación especializada en recibir mensajes de los agentes registrados y redirigir el mensaje hacia el correcto receptor. Por tanto, cualquier applet puede comunicarse con otros agentes aunque los navegadores tengan activas las restricciones de seguridad (ya que el applet sólo se conectará con su servidor). Además, es posible la comunicación asíncrona entre agentes, puesto que el AMR actúa de forma parecida al servidor de correo; es decir, espera notificación de recepción del mensaje. Si no lo recibe, almacena el mensaje en espera de que el agente receptor vuelva a ejecutarse. Por último, los agentes no deberán preocuparse de posibles cambios de direcciones de otros agentes. Por otra parte, utiliza protocolos basados en ASCII, con lo que los agentes podrían estar escritos en otros lenguajes, y podrían ser funcionales en cualquier entorno (siempre teniendo como fondo la comunicación TCP/IP). El inconveniente que se le achaca a JATLite es que no define ninguna metodología, ni leyes sociales.

Por otra parte, JAFMAS ("Java-based Agent Framework for Multi-Agent Systems")[**Chauhan, 1997**] son librerías que utilizan las características RMI de Java y la ingeniería del software basada en agentes (ABSE) para desarrollar herramientas de desarrollo de sistemas multiagente. El entorno pretende ayudar a los desarrolladores, tanto expertos como noveles, a estructurar sus ideas en aplicaciones concretas de agentes.

5.2.2 Capa de Conocimiento

CLIPS

CLIPS ("C Language Integrated Production System") es una herramienta para el desarrollo de sistemas expertos desarrollada en la NASA en 1985 ([**Clips, 1993**] y [**Giarratano, 1998**]). La características principales son:

- *Representación de Conocimiento*: proporciona un herramienta para manejar una amplia variedad de conocimiento, soportando programación procedural, basada en reglas y orientada a objetos.
- *Portabilidad*: está escrita en C y puede ser llevada a distintos sistemas sin cambios.
- *Integración* y *extensibilidad*: está embebido en el código y es fácilmente extensible por el usuario por medio de unos protocolos bien definidos.
- *Desarrollo interactivo*: proporciona un entorno de desarrollo interactivo.
- *Verificación* y *validación*: incluye una serie de características que soportan la verificación y validación de los sistemas expertos.
- *Documentación*: viene con una extensa documentación.

Aunque originariamente la primera metodología de CLIPS fue el encadenamiento hacia delante, se han ido incorporando nuevas posibilidades. Así soporta programación procedimental y orienta a objetos, adición dinámica de reglas, definición de estrategias para la resolución de conflictos, etc.

FuzzyCLIPS ([**FuzzyCLIPS, 1994**]) es una versión extendida de CLIPS para representar y manipular hechos y reglas difusas, posibilitando tanto el razonamiento exacto, inexacto (o fuzzy) y combinado. Maneja los dos conceptos de incertidumbre (ver apartado 1.6): posibilidad y valores de certeza.

Jess

JESS ("Java Expert System Shell") es una herramienta implementada en Java que va a permitir introducir razonamiento basado en reglas en cualquier programa desarrollado a partir del lenguaje Java ([**Friedman-Hill, 2001**] y [**Jess web, 2001**]). Básicamente, Jess es un intérprete de un lenguaje de reglas similar a CLIPS y el núcleo del lenguaje Jess es compatible con CLIPS. Con Jess podemos escribir aplicaciones y applets java que tengan la capacidad de razonar por medio de conocimiento suministrado en forma de reglas. Además, Jess integra amplias capacidades para crear y utilizar objetos Java dentro de su propio entorno. De esta manera, podemos ampliar el lenguaje al estilo de CLIPS mediante el potente uso de objetos creados en Java. Jess puede ser utilizado de distintas maneras, desde la línea de comandos, en aplicaciones GUI, servlets y applets. Para poder utilizarlo de manera rápida y básica, se puede iniciar el intérprete de comandos tecleando:

```
Java jess.Main
```

Es importante resaltar que dicha llamada debe hacerse desde el directorio padre de Jess; esto también es válido para las llamadas a los paquetes de Jess.

Jess es una librería para el programador que sirve como intérprete para el *lenguaje Jess* (muy similar al CLIPS). De esta forma podemos trabajar con programas escritos en CLIPS desde Java, o escribir nuestros nuevos programas de razonamiento utilizando el lenguaje Jess. El motor de inferencia de Jess es el algoritmo Rete ([**Forgy, 1982**]) de emparejamiento de reglas en una base de conocimiento. Jess permite la creación de objetos Java mediante la función `new` y llamadas a métodos de los objetos llamando a `call`. Además, nos permite interactuar con Java Beans, actuando directamente sobre sus parámetros (según métodos getXX y setXX). Internamente, nos permite crear paquetes de funciones en Java (mediante la interfaz *Userfunction*) para posteriormente poder llamar a estas funciones desde Jess como si fueran funciones propias del sistema. También nos permite redirigir streams de entrada y salida y enviar/recibir objetos básicos (cadenas, números, hechos, etc.) de un lenguaje a otro. Por último, permite definir GUI's básicas de Java dentro de Jess para una completa interactuación entre ambos lenguajes.

Una pequeña descripción de la sintaxis de estos lenguajes la podemos encontrar en el Apéndice B.1 (página 163).

5.3 Arquitectura del Sistema

El sistema de *Multiagentes Inteligentes basado en Java* (JAI) ([**Romero, 2000**]), desarrollado en el departamento de Ciencia de la Computación e Inteligencia Artificial de la Universidad de Alicante, nace como una arquitectura básica de comunicación entre agentes capaces de poseer conocimiento. Tal y como hemos visto en los primeros puntos, actualmente se habla de sistemas monoagentes (algunas herramientas tampoco tendrían que recibir el nombre de agentes), o bien de sistemas donde lo importante es la comunicación, pero que adolecen de utilizar agentes demasiado estáticos, en lo que se refiere al aprendizaje de los mismos; es decir, son agentes de un único razonamiento. En el sistema JAI se hace hincapié en dos puntos importantes:

1. Por un lado, la necesidad de seguridad y estandarización en las comunicaciones. Seguridad en cuanto a no dejar pérdidas de mensajes entre agentes; esto es, un agente que momentáneamente deja de funcionar no implica que pierda los mensajes que el resto le envían. Podría significar que dicho agente pierde conocimiento respecto a los demás por haber perdido ciertos mensajes durante su estado de inactividad. Hoy en día, los sistemas multiagente se pierden en un cúmulo de protocolos más o menos útiles que impiden que tales sistemas puedan desarrollarse con un cierto éxito. No obstante, el protocolo que más adeptos ha encontrado es el KQML (ver apartado 4.4.2, página 68). En este sentido, el sistema JAI ofrece un cierto entendimiento del protocolo, y lo enriquece mediante la serialización de objetos en Java, pudiendo comunicarse bien mediante cadenas de texto en perfecto KQML, como mediante mensajes objetos Java, que guardan de manera intrínseca la portabilidad del protocolo KQML.

2. Un segundo punto necesario para que el sistema funcione eficientemente es el relacionado con el conocimiento. Para ello, se ha utilizado un potente sistema basado en reglas, el Jess (ver apartado 5.2.2, página 82), que une ventajas de un lenguaje basado en reglas como CLIPS con ventajas en el uso de objetos nativos Java. Esta unión de fuerzas nos permitirá crear agentes capaces de razonar, aprender y crear sus propios entornos gráficos sin que sea necesario un esfuerzo superfluo por parte del programador.

El sistema JAI pretende ser la base de un sistema multiagente desarrollado de manera que tenga una fuerte proactividad y sepa evolucionar según los cambios que el entorno introduce. Esto significa que, por encima de conocimientos del agente, es necesario que éste sea capaz de

negociar con el resto del sistema, de buscar ayuda en caso de no conocer una respuesta, de adquirir conocimiento que otros agentes pueden comunicarle. Con un cierto comportamiento cívico donde los agentes sean capaces de negociar soluciones y de compartir conocimiento, el sistema será capaz de evolucionar "per se" hacia un movimiento mecánico adecuado a la evolución del entorno.

5.3.1 Arquitectura de Capas

El sistema se estructura en tres capas, claramente diferenciadas entre sí:

1. Capa de agentes
2. Capa de comunicación a nivel lógico
3. Capa de comunicación a nivel físico

El sistema trabaja a nivel TCP/IP, con lo que necesita de un soporte físico que controle el paso de mensajes, los agentes activos (o inactivos) del sistema, etc. Para ello, existe un servidor físico que nos informará de dónde se encuentra (físicamente) un agente, a dónde debemos dirigir nuestros mensajes para que éstos lleguen a buen puerto. Pero esto no es suficiente. Necesitamos saber qué agentes son expertos en una determinada materia; es necesario llevar un control de qué agentes sirven exclusivamente para negociar, cuáles controlan un subconjunto de agentes con conocimiento afín, etc. En resumen, alguien debe proporcionarnos cualidades específicas de un agente, o un conjunto de ellos. Esto lo mantendrán agentes exclusivamente dedicados a recabar tal información, que además, por su integración en el grupo, podrán aplicar razonamientos a tal información si así lo consideran oportuno. Podrían ser capaces, por ejemplo, de notar que un cierto agente supervisor tiene bajo su control a demasiados subagentes, y podría ser capaz de redirigir a algunos de estos subagentes hacia otro supervisor que no tenga tanta carga de trabajo. Este desarrollo lo integran los agentes de la capa de comunicación a nivel lógico. Por último, en la parte más alta del sistema cohabitan distintos agentes, cada uno con su propio conocimiento, o supervisores de agentes con conocimiento afín, de manera que saben dónde dirigirse, sin necesidad de buscar un agente específico para cada problema.

5.3.2 Componentes

Hemos creado el paquete java *JIAgentes* (Agentes Inteligentes en Java) que agrupa las clases que necesitamos para implementar los agentes in-

teligentes. En la figura 5.4 (página 95) podemos ver el diagrama UML ("Unified Modeling Language") del paquete *JIAgentes* que nos facilita la visión de la estructura de clases e interfaces del mismo. En la figura 5.2 vemos el diagrama de clases UML, entre las que se encuentran las dos clases principales del sistema, la clase *ServerAgente* y la clase *JKAgente*, las cuales pasamos a describir.

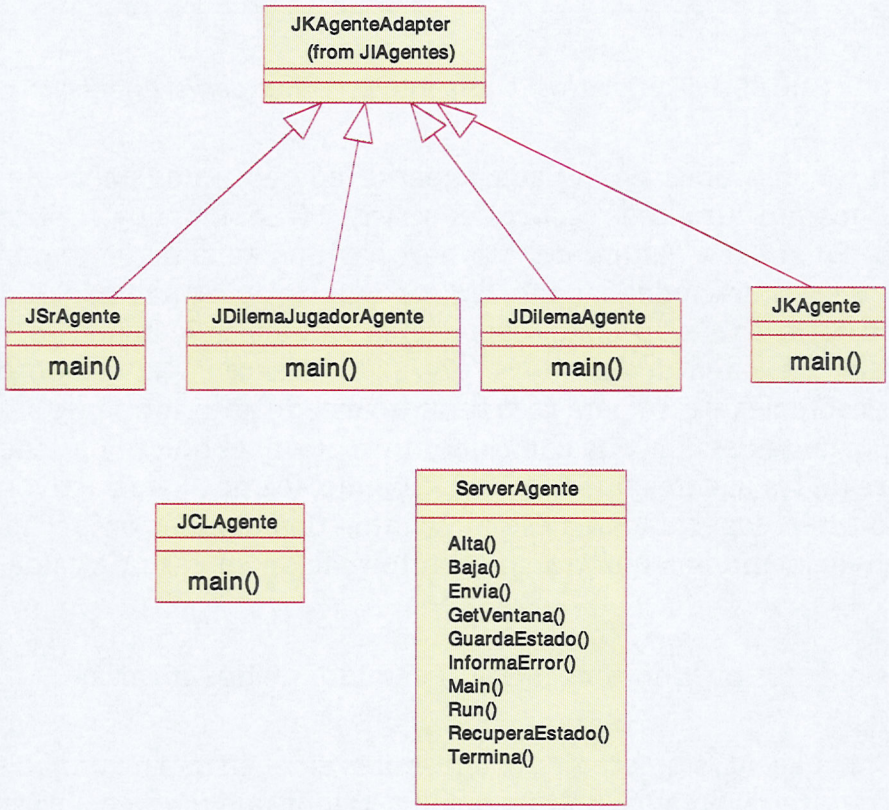


Figura 5.2: Diagrama UML del resto de Clases del Sistema

El Servidor: *ServerAgente*

La clase *ServerAgente* se encarga de encapsular las comunicaciones a nivel físico a través de TCP/IP. Su uso es el siguiente (la tabla 5.1 contiene la descripción de los parámetros):

```
Java ServerAgente [-texto] [-p <Número de puerto>]
```

Por defecto, el servidor tendrá apariencia gráfica y se conectará como `localhost` en el puerto `1234`. Cuando se inicia el servidor con entorno gráfico, tiene la apariencia de la figura 6.1 (a) (página 103). Como

`-texto`	si se desea ejecutar bajo una consola en modo texto.
`-p <Número de Puerto>`	para especificar un puerto distinto al utilizado por defecto (`1234`). Si no es un número válido, el servidor no se ejecutará.

Tabla 5.1: Parámetros del Agente Servidor (*ServerAgente*)

podemos comprobar, en la parte superior aparecen dos líneas de texto donde nos muestra la dirección del servidor (`localhost`) y el puerto de acceso. En la parte central del servidor hay una zona de texto donde el servidor escribirá todas las incidencias que vayan ocurriendo. En caso de ejecutarlo en modo texto, éstas se escribirán en la salida estándar. Por último, existen dos botones, uno para almacenar el estado de los agentes actuales y otro para recuperar un estado anteriormente salvado. Es importante detallar que con esta información, lo que se obtiene es el nombre de los agentes que tengan recipiente, los que estén activos y los que no estén conectados en ese momento. Una ampliación del sistema sería crear demonios que traten tal información en caso de caída de la red.

Los agentes pueden conectarse al servidor de dos formas:

- Si los agentes quieren estar presentes en el sistema, aunque se encuentren inactivos, notificarán al servidor que quieren abrir un recipiente donde almacenar los mensajes que el resto de agentes envíen. De esta manera, el agente puede detener su ejecución momentáneamente sin perder información sobre lo que ocurra en el entorno. Cuando vuelva a activarse, recogerá todos los mensajes en el orden en que los hubiera recibido. En este caso, el servidor actúa como un servidor de correo, utilizando un protocolo seguro que impide el borrado de los mensajes mientras el agente destino no los reciba.
- Si el agente desea aparecer de forma temporal, puede conectarse sin necesidad de crear ningún recipiente, con lo que no se almacenarán los mensajes que recibiría si estuviera dormido.

Por supuesto, el servidor también permite dos formas de desconexión, con lo que el agente que se dé de baja en el sistema puede avisar al servidor de que quiere borrar el recipiente, o bien que simplemente se da de baja temporalmente.

El Agente Especializado: *JKAgente*

El corazón del sistema es el *JKAgente*, el agente con conocimiento. Su uso es el siguiente (la tabla 5.2 de la página 94 contiene la descripción de los parámetros):

```
Java JKAgente [-texto] [-s <Nombre de Servidor>]
              [-p <Número de Puerto>] [-r]
              [-f <Nombre de Fichero>] [-n <Nombre de Agente>]
```

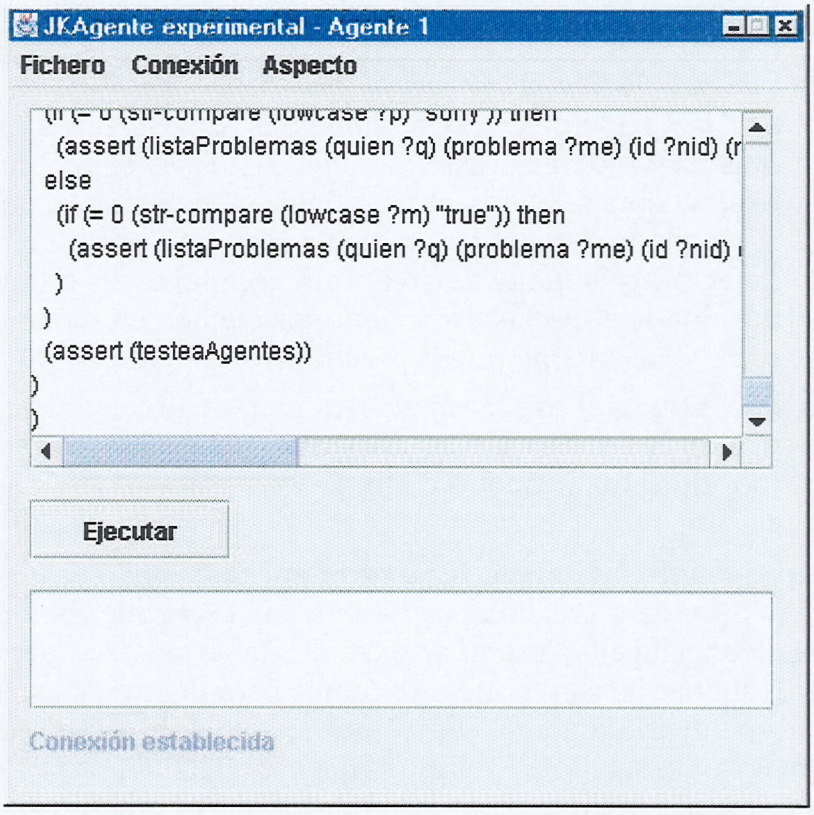


Figura 5.3: Ventana de un Agente Especializado (*JKAgente*)

Si ejecutamos el JKAgente, se abrirá la ventana de la figura 5.3, en la que podemos observar cuatro zonas. En la parte superior tiene tres menús, que más tarde pasaremos a desglosar. En la parte central hay un cuadro de texto donde se mostrará el programa en CLIPS que conforma su conocimiento. Debajo hay un botón que sirve para iniciar la ejecución del programa en CLIPS. Y, por último, en la parte inferior hay otro cuadro de texto donde el sistema escribirá los mensajes correspondientes a la evolución de la ejecución. En principio, está escrito el nombre del agente por defecto. Debajo de la zona de texto del sistema existe una etiqueta

que informa sobre si el agente está conectado a algún servidor o no. Las distintas opciones de los menús son las siguientes:

- En el menú **fichero** nos encontramos con las opciones de abrir un determinado fichero (por defecto, de extensión `.clp`), que es el archivo de razonamiento del agente. También se nos permite guardar el contenido del cuadro de texto del programa en CLIPS en un archivo. Podemos cambiar el nombre del agente, y podemos limpiar los cuadros de texto (cuidado con esta opción, ya que al borrar el archivo de CLIPS también eliminamos el razonamiento del agente). Por último, podemos cerrar el agente mediante la opción de salir.
- Con la opción de **conexión** podemos conectarnos a cualquier servidor. En el caso de conectar, el agente preguntará el nombre del servidor y el número de puerto. Además, si la opción de guardar en el servidor está activada (por defecto no lo está), se creará un recipiente en el servidor donde almacenar los mensajes que, en un momento dado, el agente no pueda recibir directamente.
- Por último, el menú aspecto es meramente estético, ya que nos permite cambiar el aspecto del agente, en metal (por defecto), windows o motif.

Existen algunos componentes más (como hemos visto en el esquema de la figura 5.2), pero creados por motivos de experimentación, con lo que se explicarán en el apartado de ejemplos (apartado 6.2, página 101). Por ahora, este par de clases son suficientes para definir el sistema. Hay que notar que el JKAgente no define por sí solo un agente incluido en la capa lógica o de agentes. Es importante este detalle porque lo que va a diferenciar a los agentes no es la implementación que tengan sino la programación CLIPS, es decir, su razonamiento. Cualquier agente puede comportarse de distintas maneras en distintos momentos, lo que da como resultado un sistema realmente dinámico con respecto a los cambios del entorno. Como ejemplo, se podría describir un agente que, a falta de un conocimiento, actuara como moderador entre otros posibles agentes y dejara por un momento su papel de agente básico de conocimiento para adoptar tareas más supervisoras.

5.3.3 Instrucciones Generales para Programar los Agentes

Para que el JKAgente pueda funcionar mediante los protocolos que ofrece, es necesario seguir unas sencillas pautas para su correcto funciona-

miento. En el momento en que un JKAgente se conecta al servidor inicializa los canales de entrada/salida en las variables `inStream` y `outStream`, accesibles desde CLIPS mediante la instrucción `fetch` (de ahora en adelante, a estas variables las denominaremos *variables fetch*). Es importante notar que si se ejecuta el razonamiento antes de conectarse, las variables contendrán un valor `nil`. El JKAgente guarda automáticamente la variable global `nombreAgente` para almacenar el nombre del Agente:

```
(defglobal ?*nombreAgente* = "Nombre del agente")
```

Si se cambia de nombre al agente, hay que limpiar el texto (esto es, inicializar el agente); de otra manera, esta variable global no cambia (aunque nosotros sí hayamos cambiado el nombre del agente). En principio, esto será suficiente para la mayoría de las aplicaciones (normalmente, los agentes no van a cambiar de nombre por lo que significaría con respecto al servidor, al resto de agentes, etc.). De todas maneras, para poder globalizar el uso de los agentes, se ha almacenado una variable fetch llamada `nombreAgente` que se inicializa cada vez que se cambia el nombre del agente. Hay que notar, sin embargo, que devuelve un `java.lang.String`, con lo que para usar este nombre habría que hacer algo parecido a lo siguiente:

```
...
(?sauxi = (fetch nombreAgente))
...
(printout t (?sauxi toString) crlf)
...
```

Si el agente recibe un mensaje, éste se almacena en la variable fetch `ultimoMensaje`, y el intérprete de CLIPS ejecutará la instrucción que nos permite tratar el mensaje recibido:

```
(assert (recibidoMensaje))
```

El objeto (Java) que nos va a permitir enviar y comprender mensajes del sistema es el Enviador, en cuyo constructor es necesario pasarle el canal de salida del agente. Como norma general, en el programa de CLIPS escribiremos la siguiente instrucción:

```
(defglobal ?*enviador* = (new Enviador (fetch outStream)))
```

Para enviar un mensaje, la clase Enviador tiene varios métodos, dife-

rentes entre sí por el número de parámetros:

```
envia(JIAMensaje mensaje)
envia(performativa, quien, aQuien, mensaje)
envia(performativa, quien, aQuien, mensaje, ontología)
envia(performativa, quien, aQuien, mensaje, ontología, idReplica,
      PorReplicaDeId)
```

Salvo en el primer método, en el resto los parámetros son de tipo cadena. En estos casos, el Enviador manda un objeto JIAMensaje. Si lo que se quiere es enviar una cadena de texto con las instrucciones necesarias para almacenar el mensaje en CLIPS, se deben utilizar los métodos:

```
enviaTexto(performativa, quien, aQuien, mensaje)
enviaTexto(performativa, quien, aQuien, mensaje, ontología,
           idReplica, PorReplicaDeId)
```

Los parámetros son cadenas de texto. Estos dos métodos envían la siguiente cadena de texto:

```
(deftemplate JIAMensaje "mensaje JAI"
   (slot performativa)
   (slot send)
   (slot rec)
   (slot mensaje)
   (slot ontologia)
   (slot reply-with)
   (slot in-reply-to)
)
(assert (JIAMensaje (performativa "<performativa>")
                    (send "quien")
                    (rec " aQUien")
                    (mensaje "mensaje")
                    (ontologia " ontología")
                    (reply-with "idReplica")
                    (in-reply-to "PorReplicaDeId")
))
```

Para tratar los mensajes recibidos (o cambiarlo de formato) se utiliza el siguiente método:

```
convierteMensaje(JIAMensaje)
```

Que devuelve la cadena de texto:

```
(deftemplate JIAMensaje "mensaje JAI"
   (slot performativa)
   (slot send)
   (slot rec)
   (slot mensaje)
   (slot ontologia)
   (slot reply-with)
   (slot in-reply-to)
)
(assert (JIAMensaje (performativa "<performativa>")
                    (send "quien")
                    (rec " aQUien")
                    (mensaje "mensaje")
                    (ontologia " ontología")
                    (reply-with "idReplica")
                    (in-reply-to "PorReplicaDeId")
))
```

Si lo que se desea es convertir un JIAMensaje en una cadena en el protocolo estándar KQML, se puede utilizar el método:

```
convierteKQML(JIAMensaje)
```

Que devuelve una cadena de texto:

```
(<performativa>
:sender <quien>
:receiver <aQuien>
:ontology <ontología>
:reply-with <idReplica>
:in-reply-to <porReplicaDeId>
)
```

Los campos `content` y `language` se dejan sin especificar para que sea el programador del agente el que los rellene según sus necesidades.

Como resumen vamos a enumerar los pasos mínimos que debe realizar un agente:

1. Primero, iniciamos el motor de razonamiento, cargamos el archivo de definiciones globales (`global.clp`) y creamos una instancia de la clase Enviador:

```
(reset)
(batch global.clp)
(defglobal ?*enviador* = (new Enviador (fetch outStream)))
```

2. Ahora, definimos una regla que introduzca en nuestra base de hechos los mensajes que que vayamos recibiendo. Para ello, activamos la regla siempre que ocurra `mensajeRecibido`, convertimos el mensaje en un hecho de CLIPS y lo insertamos en la base de hechos gracias a la función `eval`. Para entender este paso, hay que recordar la cadena de texto que nos devuelve el método `convierteMensaje` de la clase Enviador:

```
(defrule hayMensaje "El agente ha recibido un mensaje"
?regla1 <- (recibidoMensaje)
=>
(retract ?regla1)
(bind ?mensaje (call ?*enviador*
                convierteMensaje(fetch ultimoMensaje)))
(printout t "recibido " ?mensaje crlf)
(eval ?mensaje)
(assert (pideAyuda))
)
```

3. Describimos una regla de manera que, si la performativa recibida es `achieve` (es decir, debemos intentar hacer cierto el conocimiento enviado), debemos comprobar si podemos resolver el mensaje. Para esto, hacemos una llamada a la función `eval` sobre el mensaje obtenido del JIAMensaje. Esta operación la controlamos con la función `try...catch` de CLIPS. Si este intento de evaluar el mensaje falla, deberemos responder al agente con un `SORRY`, indicativo de que no podemos hacer cierta la pregunta. En caso contrario, se le avisa con un `TELL`:

```
(defrule probarMensaje "El agente va a tratar el mensaje"
?regla0 <- (pideAyuda)
?regla1 <- (JIAMensaje (performativa ?p&:
                  (= (str-compare (lowcase ?p) " achieve") 0))
                  (send ?s) (rec ?r) (mensaje ?m) (ontologia ?o)
                  (reply-with ?w) (in-reply-to ?t))
=>
(retract ?regla0)
```

```
(retract ?regla1)
; Intentamos hacer cierto el mensaje
(printout t "Vamos a probar " ?m crlf)
(try
(eval ?m)
(call ?*enviador* envia "TELL" ?*nombreAgente* ?s "TRUE" ?o ?w?w)
catch
(call ?*enviador* envia "SORRY" ?*nombreAgente* ?s "\"\ ?o ?w ?w)
)
)
```

Esta última regla sirve como ejemplo de actuación ante un conocimiento adquirido o deseado. Y, básicamente, estas reglas y la definición global deberán realizarse siempre que se implemente un agente que escuche al resto de componentes del sistema.

`-texto`	si se desea ejecutar bajo un terminal en modo texto.
`-s <Nombre de Servidor>`	para designar el nombre de un servidor al cual va a conectarse. Por defecto, utiliza `localhost`. Si no puede conectarse, el agente no se ejecuta.
`-p <Número de Puerto>`	para especificar un puerto distinto al utilizado por defecto (`1234`). Si no es un número válido, el servidor no se ejecutará.
`-n <Nombre de Agente>`	permite dar un nombre lógico al agente. Por defecto, el JKAgente se asigna como nombre su hashCode correspondiente.
`-f <Nombre de Fichero>`	se utiliza para cargar un archivo de texto en CLIPS correspondiente al conocimiento del agente. Por defecto, el agente no tiene ningún conocimiento.
`-r`	inicia la ejecución del programa de conocimiento del agente. Si éste se ejecuta en modo texto, ejecutará de manera automática el programa que tenga almacenado en memoria obtenido por medio de algún archivo.

Tabla 5.2: Parámetros del Agentes con Conocimiento (*JKAgente*)

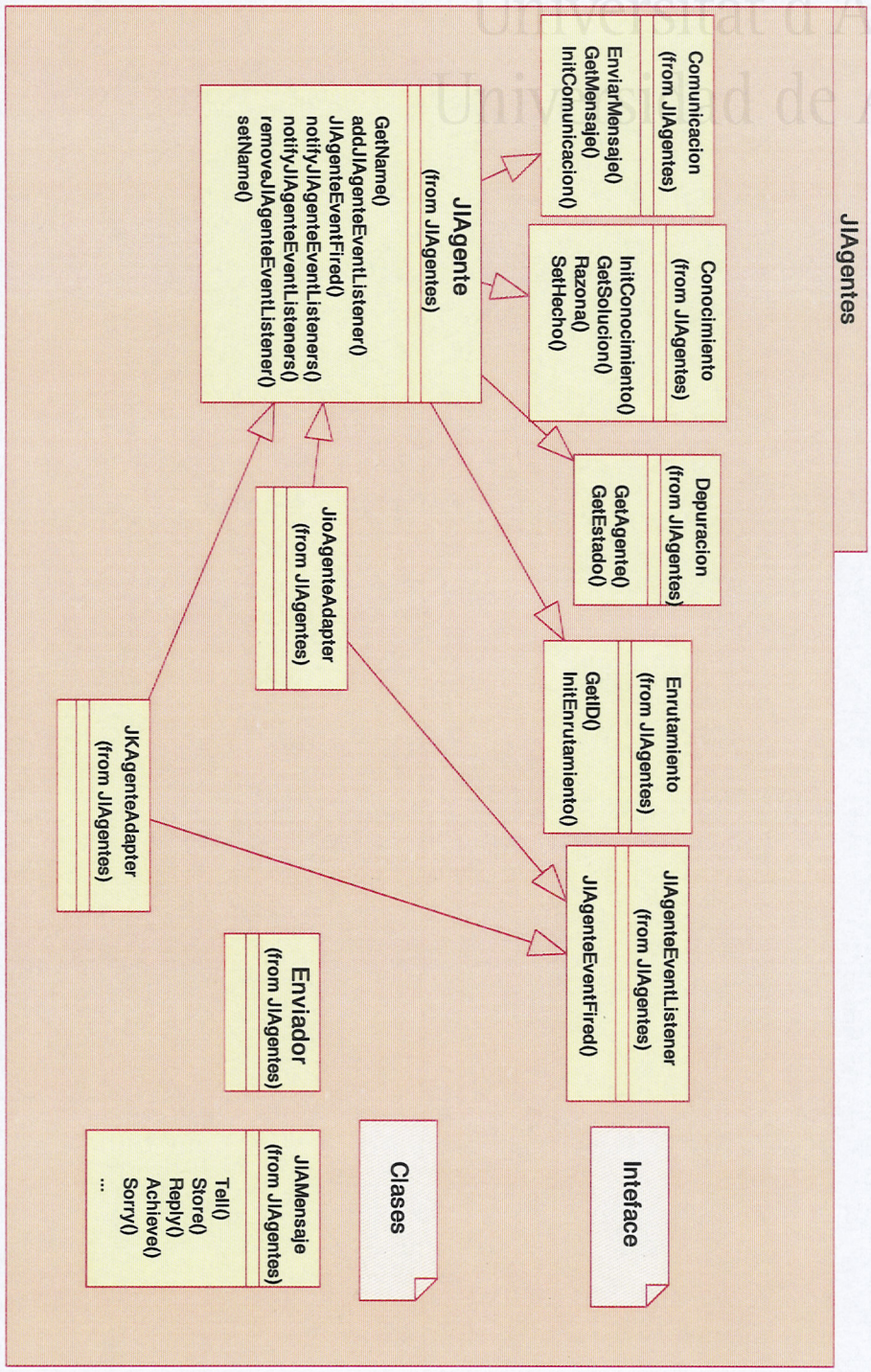


Figura 5.4: Diagrama UML del Paquete *JIAgentes*

Volver al índice/Tornar a l'índex

Capítulo 6

EXPERIMENTOS Y APLICACIONES

Estamos ahogados en información pero hambrientos de conocimiento

John Naisbitt (*Megatrends*)

Este capítulo está compuesto por dos bloques. En un primer momento vamos a analizar la aplicación de los agentes inteligente a Internet y a enumerar algunos de ellos. El segundo bloque consistirá en la descripción de los experimentos realizados, así como la valoración de los resultados obtenidos. En esta sección vamos a repasar algunos ejemplos prácticos que nos permitirán comprender cómo trabaja el sistema. En concreto, hay un primer programa que comprueba la comunicación entre un agente cliente y otro servidor; y un segundo ejemplo donde se introduce un agente que ayuda al resto a encontrar soluciones. Finalmente describimos un ejemplo más completo que juega al dilema del prisionero, cuestión seminal de la teoría de juegos.

La interacción entre la tecnología (Internet) y el comercio (transacciones económicas en Internet) supone un campo nuevo y muy interesante. Numerosos trabajos de investigación en IA están utilizando ya herramientas de teoría de juegos y economía para trabajar con las interacciones entre los agentes ([**Rosenschein and Genesereth, 1985**], [**Rosenschein, 1985**], [**Gmytrasiewicz *et al.*, 1991**] y [**Zlotkin, 1994**]).

6.1 Agentes en la Red

Internet se ha convertido en foco principal de las aplicaciones de agentes inteligentes ([**Petrie, 1996**]). La ingente cantidad de información disponible en la red así como el creciente número de usuarios conectados,

ha contribuido a la proliferación de agentes inteligentes, es decir, programas/aplicaciones que ayudan a los usuarios a realizar tareas más o menos complejas. Además, conforme crezcan las aplicaciones comerciales en Internet, estos agentes cobrarán por sus servicios y necesitaremos que nuestros agentes puedan negociar con los demás para aprovechar al máximo su utilidad. Aquí es donde entran en contacto dos grandes áreas, la Economía y la Inteligencia Artificial.

6.1.1 Los Agentes en la Práctica

Existen siete áreas donde los agentes tienen un mayor impacto:

Administración de una red de trabajo

La administración de una red de trabajo es una de las primeras áreas en las que los agentes están adquiriendo popularidad. En una empresa que tenga miles de estaciones de trabajo, un administrador no puede supervisarlo todo. Los agentes instalados en las estaciones de trabajo pueden desempeñar el papel de supervisores. Estos agentes serán los responsable de informar si hay algún problema. Pueden utilizar una serie de reglas impuestas por el administrador que les haga anticiparse a los problemas. En el futuro, los agentes de esta área podrán ayudar a localizar problemas y serán capaces de realizar tareas de administración, como distribuir las actualizaciones de software.

Correo electrónico

El correo electrónico es otra de las áreas donde los agentes están haciéndose más populares. Para algunas personas no es nada extraño recibir cientos de mensajes electrónicos en un solo día. Sin algún mecanismo que dé prioridades a este correo, o que incluso llegue a eliminar parte de él, esta gente se puede ver desbordada y sin tiempo para realizar su trabajo. Los agentes de correo electrónico existentes pueden separar por categorías y prioridades el correo que reciben, redireccionarlo a ciertas direcciones y borrar los mensajes duplicados.

Administración de la información

Una área de gran interés es la administración de la información. Los agentes son importantísimos para reducir la sobrecarga de información. Los usuarios necesitan información útil, no un montón de datos. Estos agentes necesitan comprender los diferentes formatos de los datos para registrar las bases de datos internas y externas en busca de información. Los agentes avanzados de administración de información filtrarán, condensarán y presentarán la información siguiendo una serie de reglas que le especificará el usuario.

Equipo de trabajo

Un equipo de trabajo puede ser un buen impulso para los agentes. Los agentes que pueden trabajar con equipos de trabajo pueden dirigir documentos, mandar notas recordatorias, concertar reuniones y automatizar los procesos estándar de la compañía. Un buen ejemplo ilustrativo es el proyecto Correa (COoRdinación de Recursos de Educación e investigación mediante Agentes) de la Universidad de Monterrey (México), donde los agentes informan sobre reuniones, cambios en páginas Web, etc., según la subárea en que esté inscrito el usuario.

Comercio electrónico

Los agentes dedicados al comercio electrónico pueden ayudar a los compradores a poder localizar vendedores, y viceversa. Los agentes especializados en compras pueden recorrer la Red buscando productos que reúnan una serie de características impuestas por el usuario. Una vez que recopilan la información, la pueden revisar y hacerle entonces las recomendaciones que estimen oportunas al usuario. Los usuarios más osados pueden incluso dejar que sea el propio agente el que ordene productos. Los agentes especializados en ventas pueden decidir qué información han de suministrar en respuesta a una petición.

Usuarios informáticos móviles

En unos cuantos años los agentes también desempeñarán un papel importante en la ayuda de los usuarios móviles. Los agentes que se están ejecutando en un servidor pueden seguir funcionando incluso cuando el usuario esté desconectado. Los agentes móviles condensarán la información para que se pueda transmitir por medio de un módem.

Usuarios informáticos típicos

Otra área donde los agentes pueden desempeñar un papel importante es ayudando a los usuarios a utilizar su ordenador. Lo que se espera es que el ordenador sea capaz de recordar la forma en que lo utiliza un determinado usuario y qué es lo que éste está intentando hacer. Así, el ordenador podrá hacer sugerencias referentes a una forma mejor de conseguir su propósito. El ordenador incluso puede cambiar la forma de trabajar para facilitarle el manejo al usuario. Los agentes pueden ser capaces de reconocer cuándo el usuario está repitiendo la misma combinación de teclas y crear una macro automáticamente, sugiriendo que se utilice ésta.

6.1.2 Algunos Ejemplos de Agentes en la Red

Algunos agentes que se utilizan actualmente en Internet son los siguientes:

BargainFinder es un agente de compras que permite que los usuarios puedan comparar precios entre los ocho distribuidores de CD que trabajan a través de Internet.

Eyes and Editors es un agente de compras que permite que los usuarios localicen libros que puedan resultar de interés. Guarda un registro de los libros y búsquedas que realiza el usuario y le notifica a través del correo electrónico cuándo aparecen novedades que se ajusten a su perfil.

Smart Mail es un agente de compras que mantiene puntualmente informado a los compradores sobre los productos con descuento que les pueden interesar y que estarán disponibles en breve.

NewsHound es un filtrador de noticias que busca y captura noticias basándose en los perfiles del usuario para limitar las fuentes de información.

FreeLoader es un filtro que captura sitios Web y mejora la experiencia con la red ofreciendo sugerencias que puedan ser de interés para el usuario.

PointCast Network recoge de distintas fuentes de news noticias e información basándose en los perfiles establecidos por el usuario y, a través de su interfaz, se las muestra al usuario.

CareerMart provee una serie de relaciones basándose en los conocimientos de los empleados y la base de datos de CareerMart con las ofertas de empleo disponibles, mandándolas al buzón electrónico del usuario.

Firefly hace recomendaciones musicales o cinematográficas a los miembros de la comunidad Firefly, basándose en lo que les gusta o no les gusta, además de ofrecerles la oportunidad de ponerse en contacto con otros miembros que tengan las mismas preferencias.

WebCompass crea un índice personal con la información de la Red utilizando varios buscadores, resume y clasifica los resultados, a la par que actualiza sus recursos cada vez que se utiliza.

Smart Bookmarks muestra y controla los cambios que se producen tanto en Internet como en las intranet con las que trabajamos.

6.2 Problemas Prototipo

En el campo de la Inteligencia Artificial una estrategia que produce beneficios es la de trabajar sobre versiones deliberadamente simplificadas de los fenómenos que nos interesan. Estas versiones son denominadas con humor *problemas de juguete.*

6.2.1 Comprobación del Protocolo Cliente-Servidor

Para comprobar el protocolo de conexión/desconexión entre cliente y servidor, se ha desarrollado un agente cliente, JCLAgente (figura 6.1 (b), página 103), que no tiene el motor de razonamiento ya que en este caso el agente no lo necesita. En la parte superior de la ventana, el agente tiene dos cuadros de texto donde informa del nombre del servidor y el número de puerto al que debe conectarse. Por defecto, lo intentará en `localhost` en el puerto `1234`, pero este comportamiento se puede cambiar. A continuación, una casilla nos dará la oportunidad de indicar al servidor que deseamos abrir un recipiente donde guardar los mensajes que vayamos recibiendo. Debajo aparecen otros dos cuadros de texto en los que podemos escribir el nombre del agente emisor (es decir, el que manda el mensaje; el que estamos ejecutando, en definitiva) y el nombre del agente al que enviamos el mensaje (agente receptor). En la parte central hay un área de texto donde el sistema irá mostrando los diferentes mensajes según se desarrolle la ejecución del programa. En la fila inferior de botones, encontramos la opción que nos permite conectarnos y desconectarnos al servidor. Estos comandos deberemos ejecutarlos antes de enviar cualquier mensaje. En el botón izquierdo elegiremos el mensaje que deseemos enviar mediante el botón de enviar. No todos los mensajes son performativas KQML, sino que se han ampliado para que puedan trabajar y entenderse los agentes clientes y el agente servidor del sistema. Como ya se ha dicho, el conjunto de performativas es extensible y no todas son necesarias. La tabla 6.1 muestra los mensajes que podemos mandar.

Notify	Dar de alta en la BD del servidor
Tell	Enviar mensaje
Achieve	Solicitar ayuda
End	Desconectar (se mantiene en la base de datos)
Delete	Eliminar (desaparece de la base de datos)

Tabla 6.1: Mensajes del Agente Cliente (*JCLAgente*)

Sólo `Tell` y `Achieve` son performativas KQML. De todas maneras, a

efectos del JCLAgente no efectúan ninguna operación. Para probar el protocolo, abrimos tres sesiones. En una de ellas ejecutamos el servidor, ServerAgente (figura 6.1 (a), página 103):

```
Java ServerAgente
```

Y en las dos sesiones restantes, ejecutamos el JCLAgente:

```
Java JCLAgente
```

Con lo que tendremos tres ventanas con los tres agentes: un servidor y dos clientes (figura 6.1, página 103). A un cliente le llamaremos `Agente 1` y al otro `Agente 2`. Obviamente, para el `Agente 1` el emisor es el `Agente 1` y el receptor el `Agente 2`. Y para el `Agente 2` el emisor es el `Agente 2` y el receptor el `Agente 1`. Ahora, conectamos el `Agente 1` pulsando el botón de conectar, y hacemos lo mismo con el `Agente 2`. En el Agente servidor aparece el siguiente texto:

```
Recibida nueva conexión...
Recibida nueva conexión...
```

Y en cada uno de los clientes, aparece el texto:

```
Conectando...
Me he conectado...
Conexión establecida
```

El servidor todavía no ha añadido a ninguno de los agentes en su base de datos. Para ello, seleccionamos el mensaje `Notify` (aparece por defecto) y pulsamos el botón Enviar. En el servidor aparece el texto:

```
Se ha conectado Agente 1
Se ha conectado Agente 2
```

Y en cada uno de los clientes:

```
Enviando Notify
Dado de alta en faraon
```

Si además comprobamos el directorio donde se ha ejecutado el servidor, veremos que se han creado dos directorios llamados `Agente 1` y

Agente 2, en principio vacíos. Desde el Agente 1, vamos a mandar un mensaje al Agente 2. Para ello, en la ventana del Agente 1 seleccionamos el mensaje Tell y pulsamos el botón Enviar. En el servidor, aparece:

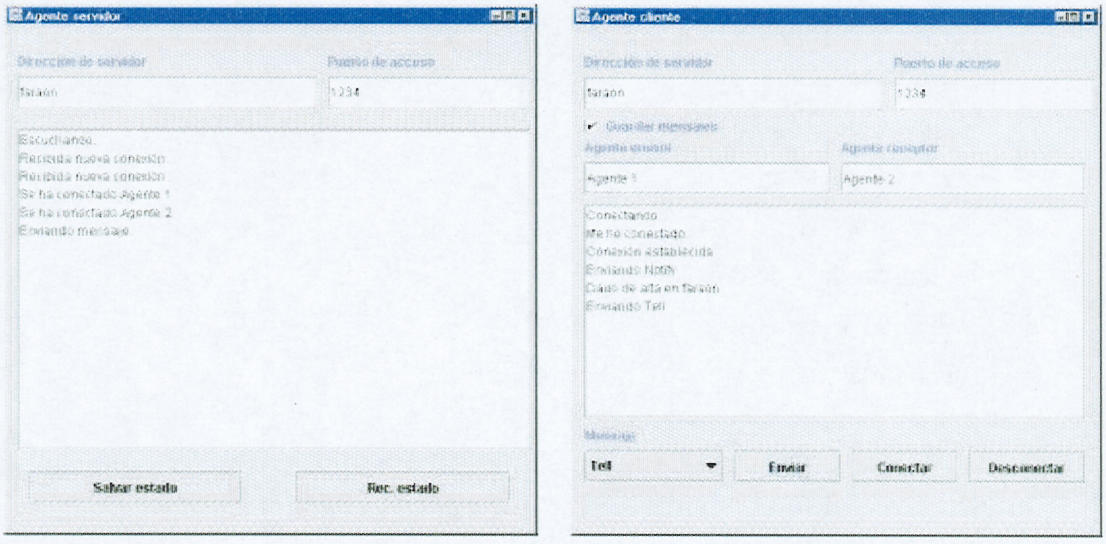
```
Enviando mensaje...
```

En el Agente 1:

```
Enviando Tell
```

Por su parte, el Agente 2 escribe:

```
Recibido un Tell
```



(a) Agente Servidor (b) Agente Cliente

Figura 6.1: Ventanas Agentes del Sistema: *ServerAgente* y *JCLAgente*

Ahora vamos a desconectar al Agente 2. Para ello enviamos un End y nos desconectamos. En el servidor aparece:

```
Se desconecta Agente 2
```

Y en el Agente 2 aparece el texto:

```
Desconectando...
Conexión cerrada
```

Si ahora mandamos un `Tell` con el `Agente 1`, aparentemente no ocurre nada. Ni siquiera el servidor escribe que se haya enviado el mensaje. No obstante, si miramos el directorio `Agente 2`, aparece un fichero de nombre `Agente 20`. Los nombres de los mensajes los renombra el servidor como `<nombreAgente0>`, `<nombreAgente1>`, y así sucesivamente. A continuación mandamos un `Achieve` y otro `Tell`. En el directorio ya tenemos los ficheros `Agente 20`, `Agente 21` y `Agente 22`. También es cierto que el servidor ha enviado mensajes `Store` al `Agente 1`, indicando a éste que sus mensajes los ha almacenado en el recipiente del `Agente 2`. Si ahora mandamos un `Notify` con el `Agente 2`, en el servidor aparece:

```
Se ha conectado Agente 2
Agente 2 | recipiente:true | conectado:true
Recuperando Agente 20
Enviando mensaje...
1 enviada
Recuperando Agente 21
Enviando mensaje...
2 enviada
Recuperando Agente 22
Enviando mensaje...
1 enviada
```

Vemos que efectivamente ha ido mandando los distintos mensajes en el orden recibido al `Agente 2`. Por otra parte, el `Agente 2` escribe:

```
Conexión establecida
Enviando Notify
Mensaje Tell de Agente 1
Mensaje Achieve de Agente 1
Mensaje Tell de Agente 1
Dado de alta en faraon
```

Y si miramos el directorio `Agente 2`, comprobaremos que está vacío. Vamos a eliminar a los agentes clientes. Para ello, mandamos un `Delete`. El servidor nos informa:

```
Se elimina Agente 1
```

```
Se elimina Agente 2
```

Podemos comprobar que los directorios recipientes han desaparecido. Desconectamos los agentes clientes y cerramos el servidor. Con esta prueba, hemos verificado que el servidor funciona perfectamente. En la figura 6.4 (página 119) podemos ver el diagrama UML de secuencias de este ejemplo, en el que se aprecia gráficamente y de forma más clara la ordenación temporal de los mensajes y la interacción entre los agentes.

En resumen, el protocolo utilizado es el siguiente:

1. El agente cliente envía un mensaje con performativa `Notify` (recordar que no es una performativa KQML en sí, sino que se utiliza internamente para asegurar el buen funcionamiento del protocolo).
2. El servidor comprueba si el agente tiene un recipiente. Si es así, comienza a enviar mensajes.
3. El cliente, por cada mensaje recibido, debe enviar al servidor una performativa `Notify`, con lo que el servidor podrá verificar que ha recibido el mensaje y, por tanto, eliminarlo del recipiente.
4. Cuando no quedan más mensajes, el servidor envía un `Notify`, de manera que el agente sabe que no hay más mensajes y que está conectado.
5. Si un agente envía un mensaje a otro agente desconectado, pero con recipiente, el servidor le responde con un `Store`, para indicarle esta situación.
6. Si un agente envía un mensaje a un nombre que no existe, el servidor le responde con un `NoAgent`, para indicarle esta situación.
7. Si el cliente envía un mensaje `End`, el servidor le dará por desconectado pero mantendrá un recipiente. Si, por el contrario, envía un `Delete`, el servidor elimina el recipiente y desaparece de su base de datos el agente.

6.2.2 Agente Matchmaker

La tarea principal asignada a este tipo de agente (ver 4.5.4, página 75) consiste en buscar a un agente que posea el conocimiento suficiente para resolver un problema que otro agente le ha preguntado. El matchmaker o facilitador no resuelve el problema, simplemente actúa de mediador entre los agentes implicados. Como siempre, necesitamos el servidor en funcionamiento. Esta vez lo mandamos en modo texto:

```
Java ServerAgente -texto
```

Vamos a utilizar a tres agentes especializados en una determinada ontología llamados `Prueba2.1`, `Prueba 2.2` y `Prueba 2.3`:

```
Java JKAgente -n "Prueba 2.1" -f prueba2.1.clp -s localhost -p 1234 -r
Java JKAgente -n "Prueba 2.2" -f prueba2.2.clp -s localhost -p 1234 -r
Java JKAgente -n "Prueba 2.3" -f prueba2.3.clp -s localhost -p 1234 -r
```

De estos tres, solamente `Prueba 2.2` contendrá la solución al problema enviado. Necesitamos al organizador, que va a ser el agente encargado de mantener listas de agentes relacionados con una determinada ontología, si están conectados, etc. Va a realizar, en definitiva, las tareas propias del nivel lógico del sistema:

```
Java JKAgente -n " organizador" -f organizador.clp -s localhost -p 1234 -r
```

Por supuesto, el agente facilitador también debe funcionar. Tiene como nombre `Agente 1`:

```
Java JKAgente -n "Agente 1" -f matchmaker.clp -s localhost -p 1234 -r
```

Por último, necesitamos el agente que pida ayuda al facilitador; es el `Agente 2`:

```
Java JKAgente -n "Agente 2" -f prueba.clp -s localhost -p 1234 -r
```

Para simplificar la simulación, el organizador incluye automáticamente tres ontologías distintas con un número cualquiera de agentes conocedores de la materia (fichero `organizador.clp`):

```
(assert (nuevoAgente (nombreOnt "Prueba") (nombreAgente "Prueba 1.1")))
(assert (nuevoAgente (nombreOnt "Prueba") (nombreAgente "Prueba 1.2")))
(assert (nuevoAgente (nombreOnt "Prueba") (nombreAgente "Prueba 1.3")))
(assert (nuevoAgente (nombreOnt "Prueba2") (nombreAgente "Prueba 2.1")))
(assert (nuevoAgente (nombreOnt "Prueba2") (nombreAgente "Prueba 2.2")))
(assert (nuevoAgente (nombreOnt "Prueba2") (nombreAgente "Prueba 2.3")))
(assert (nuevoAgente (nombreOnt "Prueba3") (nombreAgente "Prueba 3.1")))
(assert (nuevoAgente (nombreOnt "Prueba3") (nombreAgente "Prueba 3.2")))
```

El proceso que se sigue es el siguiente: el `Agente 2` envía una solici-

tud de ayuda (Achieve) al Agente 1 (el facilitador) sobre un problema de ontología Prueba2. El facilitador recibe:

```
(assert (JIAMensaje (performativa "ACHIEVE")
                    (send "Agente 2")
                    (rec "Agente 1")
                    (mensaje "(assert (problema (duda A)))")
                    (ontologia "Prueba2")
                    (reply-with )
                    (in-reply-to )

))
```

El Agente 1 pide al organizador una lista de los agentes que conocen la ontología Prueba2:

```
(assert (JIAMensaje (performativa "TELL")
                    (send "Agente 1")
                    (rec " organizador")
                    (mensaje (assert (pideLista
                                      (aQuien "Agente 1")
                                      (nombreOnt "Prueba2")
                                      (id1))))
                    (ontologia )
                    (reply-with )
                    (in-reply-to )

))
```

El organizador le devuelve un mensaje con la lista pertinente:

```
(assert (JIAMensaje (performativa "REPLY")
                    (send " organizador")
                    (rec "Agente 1")
                    (mensaje (assert (listaAgentes (lista
                                      (create$ "Prueba 2.3"
                                               "Prueba 2.2"
                                               "Prueba 2.1" ))
                                      (id 1))))
                    (ontologia )
                    (reply-with )
                    (in-reply-to )

))
```

Una vez que el Agente 1 recibe la lista de agentes, puede realizar una rueda de consultas. Primero pregunta a Prueba 2.3, que intenta hacer

cierto el mensaje:

```
(assert (JIAMensaje (performativa "ACHIEVE")
                    (send "Agente 1")
                    (rec "Prueba 2.3")
                    (mensaje "(assert (problema (duda A)))")
                    (ontologia "Prueba2")
                    (reply-with "Problema1")
                    (in-reply-to )

))
```

Pero no puede resolverlo, así que le devuelve un `sorry` al `Agente 1`:

```
(assert (JIAMensaje (performativa "SORRY")
                    (send "Prueba 2.3")
                    (rec "Agente 1")
                    (mensaje )
                    (ontologia "Prueba2")
                    (reply-with "Problema1")
                    (in-reply-to "Problema1")

))
```

Como ya ha recibido la respuesta, el `Agente 1` pregunta al siguiente agente de la lista, `Prueba 2.2`:

```
(assert (JIAMensaje (performativa "ACHIEVE")
                    (send "Agente 1")
                    (rec "Prueba 2.2")
                    (mensaje "(assert (problema (duda A)))")
                    (ontologia "Prueba2")
                    (reply-with "Problema1")
                    (in-reply-to )

))
```

El agente `Prueba 2.2` sí que tiene el conocimiento necesario para solucionar el problema, con lo que responde afirmativamente:

```
(assert (JIAMensaje (performativa "TELL")
                    (send "Prueba 2.2")
                    (rec "Agente 1")
                    (mensaje TRUE)
                    (ontologia "Prueba2")
                    (reply-with "Problema1")
                    (in-reply-to "Problema1")
```

```
))
```

Por último, el Agente 1 pregunta a Prueba 2.1, obteniendo una respuesta negativa:

```
(assert (JIAMensaje (performativa "ACHIEVE")
                    (send "Agente 1")
                    (rec "Prueba 2.1")
                    (mensaje "(assert (problema (duda A)))")
                    (ontologia "Prueba2")
                    (reply-with "Problema1")
                    (in-reply-to )
))
(assert (JIAMensaje (performativa "SORRY")
                    (send "Prueba 2.1")
                    (rec "Agente 1")
                    (mensaje )
                    (ontologia "Prueba2")
                    (reply-with "Problema1")
                    (in-reply-to "Problema1")

))
```

Una vez terminada la rueda de consultas, el Agente 1 tiene una lista con los nombres de los agentes que pueden resolver el problema; en este caso sólo puede resolverlo Prueba 2.2. Para finalizar su tarea envía esta lista al agente que le ha realizado la consulta:

```
(assert (JIAMensaje (performativa "REPLY")
                    (send "Agente 1")
                    (rec "Agente 2")
                    (mensaje "(create$ "Prueba 2.2")")
                    (ontologia "Prueba2")
                    (reply-with )
                    (in-reply-to )

))
```

De esta manera, el Agente 2 puede ponerse en contacto con los agentes especializados. En la figura 6.5 (página 120) vemos el diagrama UML de secuencias de este ejemplo, en el que se aprecia gráficamente y de forma más clara la ordenación temporal de los mensajes y la interacción entre los agentes.

Razonamientos en la simulación del facilitador

El conocimiento (razonamiento) lo tenemos en los ficheros `.clp`. En el caso de `Prueba 2.1`, `Prueba 2.2` y `Prueba 2.3` sólo existen dos reglas, una para recibir un mensaje y otra para intentar hacerlo cierto. `Prueba 2.2`, además, define una plantilla necesaria para que el problema que le plantea la simulación pueda ser resuelto (ficheros `prueba2.1.clp`, `prueba2.2.clp` y `prueba2.3.clp`).

El `Agente 2` tiene una única regla (fichero `prueba.clp`) para recibir mensajes, y envía directamente el mensaje achieve al `Agente 1`. Los agentes incluyen el conocimiento contenido en el fichero `global.clp` que se refiere a cuestiones generales. Esto sirve para evitar definiciones similares en los distintos ficheros de razonamiento. Además sirve para saber dónde se encuentran (cómo se llaman) los organizadores del sistema. De esta forma se puede cambiar el agente organizador sin necesidad de cambiar los ficheros de razonamiento. El organizador tiene un par de reglas que ayudan a almacenar los agentes (fichero `organizador.clp`). Según el valor de la variable global `?*seGuarda*`, si un agente se da de alta con una ontología desconocida, se crea una nueva lista con el nombre de la ontología, o se ignora la petición del agente. Las reglas son `nuevoAgenteConOnt` y `nuevoAgenteSinOnt`. Para comunicarse con el facilitador, tiene la regla `mandaLista` que sirve para enviar los nombres de los agentes conocedores de una ontología.

6.3 Aplicaciones en Teoría de Juegos

En 1944 von Neumann y Morgenstern publicaron el libro "Theory of Games and Economic Behavior" ([**Neumann and Morgenstern, 1944**]) en el que se aplicaban sus estudios matemáticos a la economía, la política, los asuntos exteriores y otros ámbitos. La *teoría de juegos* ([**Luce and Raiffa, 1957**], [**Davis, 1970**] y [**Maynard-Smith, 1982**]) no se refiere a "jugar" tal como lo entendemos comúnmente , sino que estudia los conflictos entre seres racionales que desconfían uno del otro[1] y la toma de decisiones en esos ambientes. Así, por ejemplo, la mejor forma de repartir un pastel en dos partes, sin que haya conflictos, es que uno corte la tarta y el otro elija el trozo. De esta manera nadie se puede quejar. La teoría de juegos busca soluciones (resultados racionales) a los juegos.

[1]Sería más adecuado utilizar el término estrategia, en lugar de juego, ya que da a entender mejor su contenido

6.3.1 El Dilema del Prisionero

En 1950 dos investigadores de RAND[2], Flood y Dresher, idearon un juego simple y desconcertante, que se conoce como el *dilema del prisionero*[3] ([**Poundstone, 1995**]). La descripción del mismo sería:

Dos hombres, acusados de infringir conjuntamente la ley, han sido confinados por la policía en habitaciones separadas sin poder intercambiarse mensajes. La situación es la siguiente:

1. El jefe de policía les propone a cada uno de ellos un pacto. Si uno de ellos testifica contra su compañero quedará libre, pero el otro será sentenciado con prisión.
2. Pero hay una trampa, ya que si ambos testifican el uno contra el otro, se condenará a los dos.
3. Al mismo tiempo, cada uno de ellos tienen sus buenas razones para creer que si ninguno confiesa, ambos tendrán una sentencia menor (la policía carece de pruebas suficientes para condenarlos por la acusación principal).

Se informa a ambos que al otro prisionero se le está haciendo la misma propuesta y se les concede un corto plazo de tiempo para que mediten la cuestión, pero en ningún momento uno podrá conocer la decisión del otro.

Condena de prisión	B rechaza propuesta	B acusa compañero
A rechaza propuesta	A: 1 año, B: 1 año	A: 3 años, B: 0 años
A acusa compañero	A: 0 años, B: 3 años	A: 2 años, B: 2 años

Tabla 6.2: Dilema del Prisionero

Aún es más interesante si analizamos el comportamiento en varios juegos del dilema. Los *dilemas del prisionero iterativos* consisten en un conjunto de dilemas del prisionero en el que cada jugador sabe que interactuará varias veces seguidas con su oponente. La mayor parte de los dilemas que se nos plantean en la realidad son iterativos. En el dilema del

[2]Organización RAND: Investigación y Desarrollo ("Research And Development")

[3]Un relato con dilema es una história que plantea una decisión difícil y que pide a la persona que escucha que lo resuelva

prisionero que se juega una sola vez existen únicamente dos opciones. Pero en un dilema iterativo del prisionero puede haber infinitas *estrategias*. En cada dilema tomaremos la decisión basándonos en lo que los otros jugadores hicieron en los dilemas anteriores. Las estrategias pueden alcanzar gran complejidad ([**Axelrod, 1984**]). La diferencia entre un dilema iterativo y uno de un solo lance es que en el primero existe la proyección del futuro. Vamos a comentar algunas estrategias sencillas:

Siempre cooperar Se trata de cooperar siempre. Si todos los jugadores siguen este principio, los resultados favorecen a todos, pero si el otro no coopera se estará "haciendo el primo".

Siempre desertar Se trata de desertar siempre. Es la estrategia más prudente, nadie podrá aprovecharse de nosotros, pero jamás se le da una oportunidad al contrario.

Cooperar o desertar aleatoriamente Se trata de cooperar o desertar al azar.

Cooperar y desertar alternativo Se trata de cooperar y desertar alternativamente.

Donde las dan, las toman Se trata de cooperar la primera vez y repetir la última jugada de nuestro contrincante el resto de las partidas.

Mezcla de cooperador y toma y daca Se trata de cooperar la primera vez y el resto de las partidas se hace: el 40% de las veces se coopera y el 60% restante, se coopera sólo si nuestro contrincante ha cooperado en sus dos últimas partidas.

Rencoroso total Se trata de cooperar la primera vez y todas las siguientes mientras el otro también lo haga. Si alguna vez el contrario defrauda, entonces desde ese momento se defrauda siempre.

La cuatro primeras estrategias son *ciegas* pues se fijan totalmente de antemano y no se tiene en cuenta lo que el otro jugador haya hecho. "Donde las dan, las toman" es una *estrategia amable*, ya que no se es nunca el primero en desertar, pero al mismo tiempo está a la defensiva.

Para que un juego pueda calificarse como un dilema del prisionero, sólo es necesario que los resultados de las diferentes jugadas se atengan a una clasificación determinada, con la siguiente estructura: existe un resultado *recompensa* en el caso de que haya cooperación entre ambos y un *castigo* para ambos si no hay cooperación; sin embargo ambos

codician el resultado de la *tentación* en el caso de que haya una única deserción; la tentación ofrece más ganancia que la recompensa y ambos temen "hacer el primo".

El dilema del prisionero es más que un simple juego, posee una estructura matemática concreta y es un problema que se plantea en la vida real (siempre que haya conflicto de intereses, en conflictos bélicos, ...). El dilema del prisionero es interesante porque desafía al sentido común. Al depender de la decisión del otro, lo mejor será desertar. El que deserta hace el siguiente razonamiento: mi decisión no puede haber influido en la del otro, por tanto si hemos desertado los dos he tenido suerte por haber decidido desertar, ya que de otra manera hubiese hecho el primo. El partidario de cooperar piensa que no debe uno aprovecharse del otro, y por tanto si los dos jugadores son racionales, lo mejor es cooperar ambos. ¿Qué hacer?

6.3.2 El Dilema del Prisionero en el Sistema de Agentes

El juego del dilema del prisionero es un problema relativamente sencillo para mostrar la manera de comunicarse e interactuar los agentes del sistema. El problema de representar el dilema del prisionero mediante una historia es que intervienen factores emocionales que no tienen nada que ver con el tema (código moral , sentirse mal, ...). Por ello vamos a plantearlo como un juego. Cada jugador posee un valor en puntos y cuando se encuentra con otro jugador, echan una partida entre ellos. Cada jugador trata de conseguir el máximo número de puntos. Los jugadores tienen dos posibles opciones: *cooperar* y *desertar*, pero un jugador no sabe qué opción va a elegir el otro. Los puntos ganados en cada caso se muestran en la tabla 6.3.

Jugador 1	Jugador 2	Puntos J1	Puntos J2
Coopera	Coopera	3	3
Coopera	Defrauda	0	5
Defrauda	Coopera	5	0
Defrauda	Defrauda	1	1

Tabla 6.3: Puntuación en el Juego del Dilema del Prisionero

Los experimentos se van a efectuar con un juego de diez partidas. Además, en cada simulación se puede elegir el estilo de juego de entre las estrategias vistas anteriormente. En el caso de la estrategia al azar, aplicaremos cooperar el 47% de las veces y desertar el resto. Todas estas estrategias se pueden simular fácilmente en el ordenador (ver regla CLIPS para la estrategia "donde las dan, las toman" en la página 117).

Como siempre, necesitamos la ejecución del servidor para poder funcionar en modo multiagente (lo ejecutamos en modo texto):

```
Java ServerAgente -texto
```

En este caso, también necesitamos un servidor lógico que arbitre las partidas:

```
Java JDilemaAgente
```

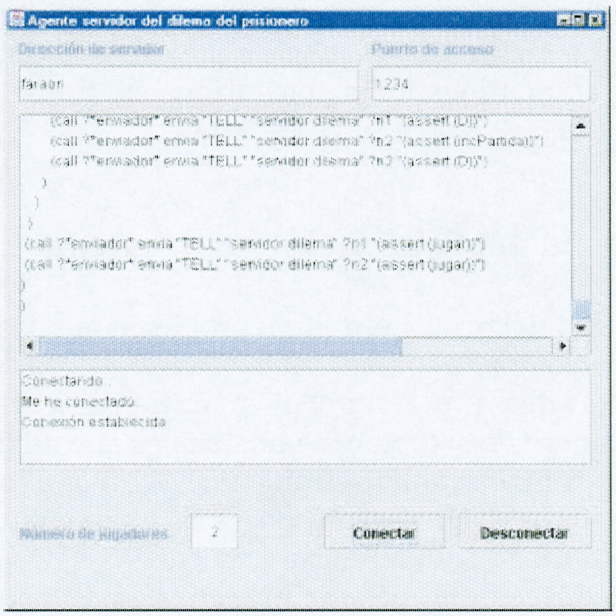


Figura 6.2: Ventana del Arbitro del Juego (*JDilemaAgente*)

Ya tenemos un agente simple que va a controlar las partidas entre los distintos agentes (figura 6.2). Al igual que el resto de agentes, tiene dos campos, uno para introducir el nombre del servidor y otro para indicar el número del puerto al cual debe conectarse. También tiene dos áreas de texto, la superior donde escribirá su razonamiento, y la inferior donde sacará los distintos mensajes del sistema. Más abajo, contiene una zona de texto donde escribir el número de jugadores implicados en el juego, y por último, dos botones que permiten conectar y desconectar el agente con el servidor físico. En el momento en que pulsemos el botón de conectar, aparecerá en el área de texto central el razonamiento utilizado por el agente. Además, nos informará de la conexión establecida. Ahora necesitamos dos jugadores. Para ello, en dos sesiones distintas, ejecutamos:

```
Java JDilemaJugadorAgente
```

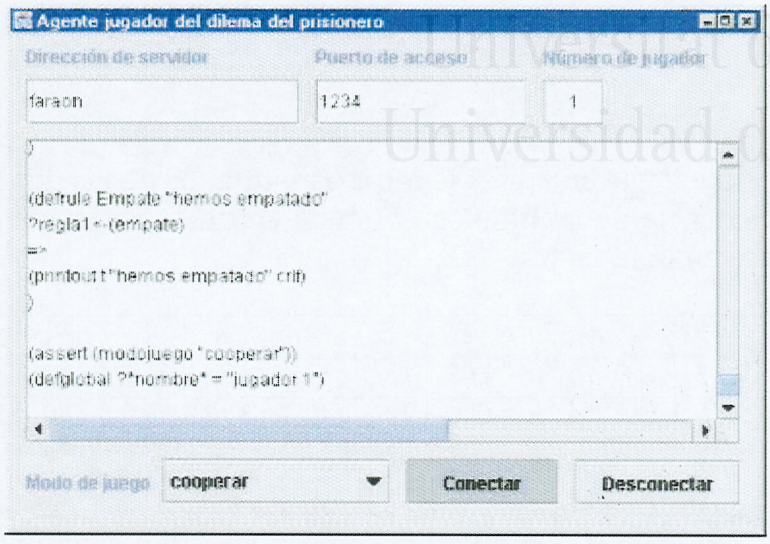


Figura 6.3: Ventana de un Jugador del Dilema del Prisionero (*JDilemaJugadorAgente*)

Con lo que nos aparecerá el agente jugador del dilema (figura 6.3). En la parte superior de la ventana, como en todos, hay un par de campos para la dirección del servidor y el puerto de acceso. Además hay un tercer cuadro de texto donde introducir el número de jugador (1,2...), para poder diferenciarlos entre ellos. En la zona central aparecerá su razonamiento y los distintos mensajes del sistema. En este agente se ha preferido una sola zona de texto para simplificar la apariencia del mismo. En la lista desplegable podemos elegir el modo de juego de entre los indicados anteriormente. Es importante elegir uno antes de conectarse, ya que será el elegido para jugar la partida. En el momento en que se elija uno, en el área de texto se explicará brevemente el modo elegido. Para hacer la prueba, elegiremos el modo "cooperar el 47%" para el `agente 1` y el modo "donde las dan las toman" para el `agente 2`. Cuando se conecta el `agente 1`, aparece su razonamiento y los mensajes

```
Dado de alta en el servidor
Mandando petición de jugar...
```

Y en el servidor se escribirá

```
1 de 2
```

Y quedará esperando que se conecten el resto de jugadores. En el momento en que se conecta el `agente 2`, la acción se dispara. En el servidor

aparece:

```
2 de 2
Podemos empezar a jugar
```


Y en los agentes van apareciendo los resultados de las distintas jugadas. La secuencia de resultados obtenida en una simulación ha sido la que aparece en la tabla 6.4.

Jugada	1	2	3	4	5	6	7	8	9	10	Ptos
Jugad.1	D 5	D 1	C 0	D 5	C 0	C 3	C 3	D 5	D 1	D 1	24
Jugad.2	C 0	D 1	D 5	C 0	D 5	C 3	C 3	C 0	D 1	D 1	19

Tabla 6.4: Simulación 1 del Dilema del Prisionero Iterativo

Como vemos, el `Agente 1` ha desertado 6 veces y colaborado 4. Y el `Agente 2` ha cooperado la primera vez y ha realizado la jugada anterior del contrincante en el resto de turnos. En este caso ha obtenido más puntos el `Agente 1`.

Si lanzamos otra simulación, en la que el `Agente 1` utiliza la estrategia "mezcla de cooperar (el 40%) y toma y daca" y la estrategia "donde las dan las toman" para el `Agente 2`, obtenemos la puntuación que aparece en la tabla 6.5. En este caso han colaborado los dos todas las veces.

Jugada	1	2	3	4	5	6	7	8	9	10	Ptos
Jugad.1	C 3	C 3	C 3	C 3	C 3	C 3	C 3	C 3	C 3	C 3	30
Jugad.2	C 3	C 3	C 3	C 3	C 3	C 3	C 3	C 3	C 3	C 3	30

Tabla 6.5: Simulación 2 del Dilema del Prisionero Iterativo

Razonamiento del servidor del juego

El agente árbitro del juego utiliza el fichero `dilema.clp` donde contiene su razonamiento. Antes, inicia la variable global `?*nj*` para guardar el número de jugadores. Inicia una variable global `?*npiv*` para guardar el número de jugadores actual, y crea un Enviador. Cada vez que un jugador se conecta, realiza un `assert` de una plantilla `masjugador` donde el propio jugador envía su nombre. El servidor responde a esta inserción mediante la regla `anyadejugador`, en la que añade una plantilla con el nombre y el identificador del agente jugador. Si ya están todos, introduce el hecho jugar, que dispara la regla `cupo` que permite disparar

a su vez a la regla `notif` que manda a los jugadores el conocimiento necesario para que realicen sus jugadas (una cada vez). En la primera jugada recibida por los agentes (comprobaremos que las jugadas provienen de agentes distintos), vamos a almacenar una plantilla combate donde guardaremos información sobre el desarrollo de la partida. Esto lo hace la regla `preparapartida`. Si, por el contrario, ya llevamos más jugadas y tenemos una plantilla de combate, actualizamos ésta mediante la regla `preparaPartidaconCombate`. Cuando hayamos introducido los datos de la jugada actual en una plantilla partida, comprobaremos quién ha ganado en la regla `juega`. Si ya hemos jugado diez partidas, se envía a cada jugador el resultado del combate. En caso contrario, se le manda la partida del otro jugador y se le invita a jugar de nuevo.

Razonamiento del cliente del juego

El cliente utiliza el fichero `jugador.clp`. Para saber el modo de juego utilizado, el agente inserta un hecho `modoJuego` con el nombre del mismo. Para guardar el desarrollo de la partida, el agente crea una variable multicampo `juego`, donde el primer campo indica el número de partida, y el resto de campos sirven para almacenar las partidas del contrario. Si hay una N, no se ha jugado todavía; C significa cooperar y D desertar. Utiliza una variable auxiliar para el rencoroso, ya que éste sólo cambiará de jugada cuando el contrario defraude. Por supuesto, crea un enviador. Cada modo de juego tiene su propia regla, que se activará cuando exista un hecho `jugar` enviado por el servidor; se disparará la correspondiente regla según la variable `modoJuego`. Por otra parte, existen las reglas `incrPartida` para incrementar el número de partida actual, y las reglas que insertan la jugada del contrario (`otroC` y `otroD`). Por último, existen tres reglas que informarán del resultado del combate (`Victoria`, `Empate` y `Derrota`).

```
(defrule jdantoman "toman y daca"
?regla1<-(jugar)
(modojuego "donde las dan, las toman")
=>
(retract ?regla1)
(printout t "A jugar donde las dan, las toman" crlf)
(if (= 0 (nth$ 1 ?*juego*)) then
     (call ?*enviador* envia "TELL" ?*nombre* "servidor dilema"
           (format nil "(assert (jugada (nombre \"%s\")
                                (jugada C)))" ?*nombre*))
```

```
else
     (if (eq C (nth$ (+ 1 (nth$ 1 ?*juego*)) ?*juego*)) then
       (call ?*enviador* envia "TELL" ?*nombre* "servidor dilema"
            (format nil "(assert (jugada (nombre \"%s\")
                                         (jugada C)))" ?*nombre*))
  else
       (call ?*enviador* envia "TELL" ?*nombre* "servidor dilema"
            (format nil "(assert (jugada (nombre \"%s\")
                                        (jugada D)))" ?*nombre*))
  )
)
)
```

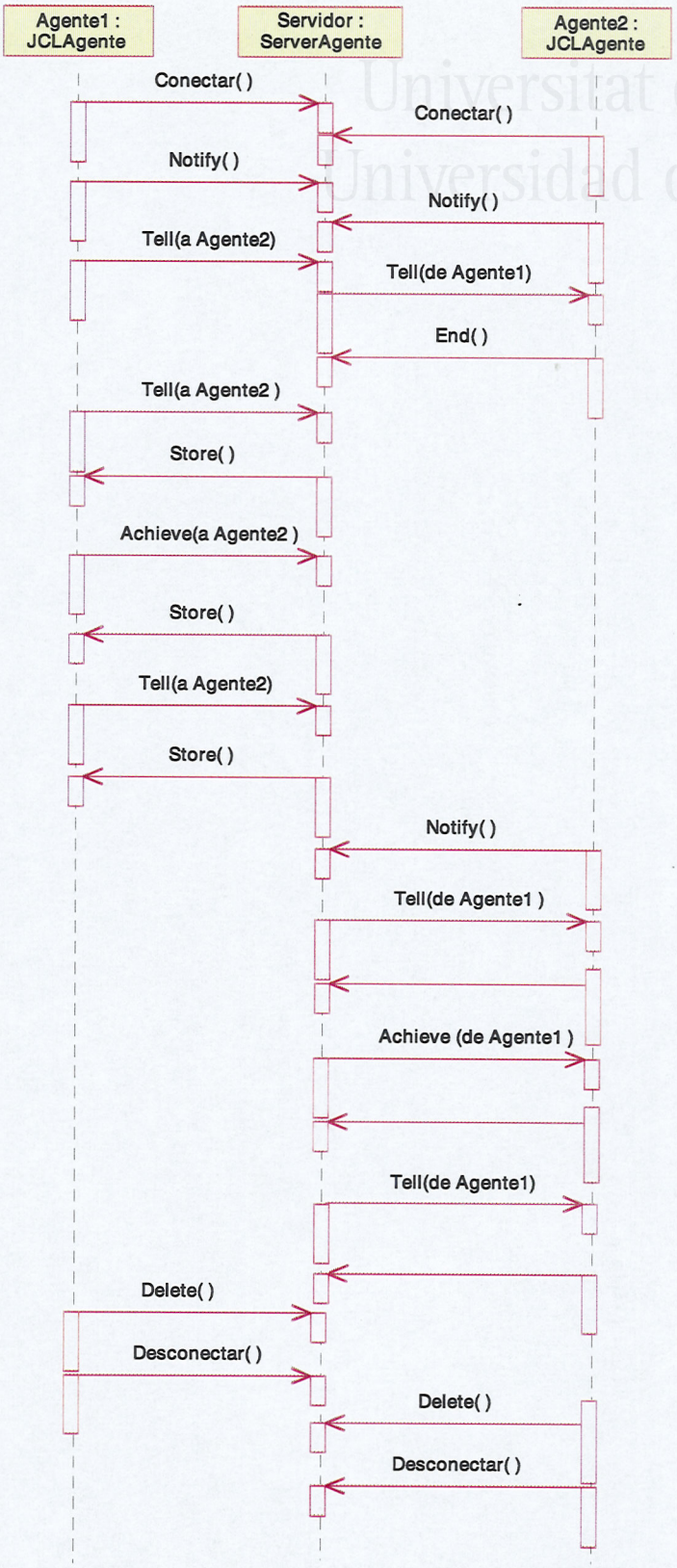


Figura 6.4: Diagrama de Secuencias del Ejemplo 1 (Protocolo Cliente-Servidor)

Volver al índice/Tornar a l'índex

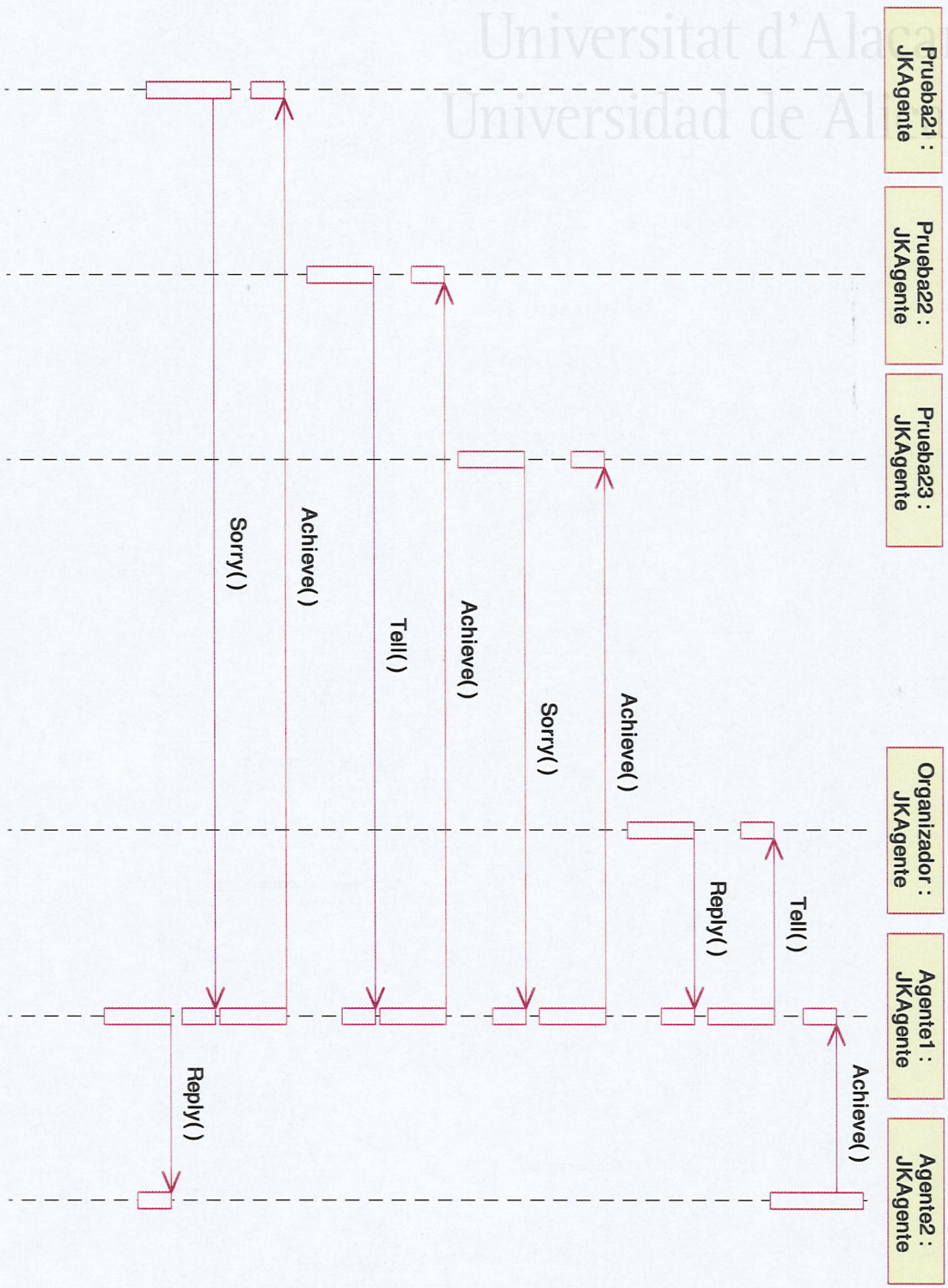


Figura 6.5: Diagrama de Secuencias del Ejemplo 2 (Agente matchmaker)

Capítulo 7

CONCLUSIONES

Después de revisar la situación actual de la representación del conocimiento, del razonamiento y de la vertiente más aplicada, los sistemas basados en conocimiento, se ha propuesto un modelo de representación y razonamiento que nos permite adecuar los avances realizados de forma "clásica" al nuevo medio de conectividad, coordinación y procesamiento basado en Internet, mediante los agentes inteligentes. Más concretamente se propone el modelado basado en agentes inteligentes, constituyendo el enlace entre el medio físico (internet junto con los procesadores y memorias distribuidas), la representación del conocimiento y los procesos cognitivos (razonamiento y deducción).

Para ello, en primer lugar se ha realizado un estudio y recopilación de información del contexto general y un estudio y análisis detallado de la situación actual, en los campos en que se basa el trabajo:

- Modelos de representación de conocimiento.
- Técnicas de razonamiento, en especial los sistemas basados en reglas.
- Agentes inteligentes y sistemas multiagente.
- Métodos de coordinación y comunicación, en situaciones tanto de colaboración como de competencia entre agentes.

Se han estudiado y analizado distintas propuestas de lenguajes y formatos, que se están convirtiendo en estándares reconocidos por especialistas en estos campos:

- Lenguaje ACL ("Agent Communication Language") y su utilización para la intercomunicación entre agentes autónomos distribuidos.

- Formato de representación KIF ("Knowledge Interchange Format"), lenguaje para el intercambio de conocimiento entre sistemas heterogéneos.

- Protocolo KQML ("Knowledge Query and Manipulation Language"), lenguaje que maneja los mensajes para permitir a un programa identificar, conectarse e intercambiar información con otros programas.

Tras ello se ha realizado un estudio de las últimas investigaciones en el campo, así como la identificación de los aspectos susceptibles de ampliación y mejora. Se ha investigado sobre los medios y las tecnologías actualmente disponibles para la construcción de sistemas multiagente. Se han evaluado distintas herramientas para su instrumentalización decidiéndonos por la utilización de Java como lenguaje genérico, Jess ("Java Expert System Shell") como motor de inferencia, lo que nos permite escribir el conocimiento en el lenguaje de Jess, cuya sintaxis está basada en CLIPS ("C Language Integrated Production System") y KQML como lenguaje y protocolo de comunicación entre los agentes.

Como consecuencia se ha aportado un modelo de Sistema de Agentes para su uso en redes, con un planteamiento de soporte formal de caracterización para la implementación de dicho sistema multiagente. Dicho sistema se caracteriza por la existencia de agentes especializados o agentes con conocimiento (JKAgentes), los cuales adquieren su conocimiento y, por tanto, su especialización al cargar e interpretar ficheros en CLIPS/Jess.

Se ha verificado la bondad del modelo por su versatilidad, ya que con una clase de objetos genérica, disponemos de múltiples y diferentes agentes con sólo proporcionarles el conocimiento adecuado.

Por la estructura que presentan, los agentes pueden desempeñar distintos roles en distintos momentos y adquirir nuevo conocimiento, al mismo tiempo que pueden compartir el conocimiento con los demás agentes.

Se han incorporado mecanismos para no perder integridad en el paso de mensajes debido a indisponibilidades temporales de algún nodo.

Las líneas que quedan abiertas y que, por tanto, serán fruto de investigaciones futuras son:

- Incorporación de criterios de capacidad de respuesta del agente incorporando medidas de certeza.

- Abordar la viabilidad de coexistencia de otros motores de inferencia de tipo declarativo.
- Abordar la movilidad de los agentes, para permitir la migración, en aras de mejorar el uso de los recursos disponibles.
- Avanzar en la coordinación del aprendizaje derivado de la interacción entre los agentes.
- Trasferir el prototipo propuesto a situaciones reales, implicando a empresas interesadas en su utilización.

Volver al índice/Tornar a l'índex

BIBLIOGRAFÍA

[**Allport, 1988**] Alan Allport. *What Concept of Consciousness?*, page 159. In Marcel and Bisiach [**1988**], 1988.

[**Anderson, 1993**] J. R. Anderson, editor. *Rules of the Mind.* Erlbaum, 1993.

[**Arbib, 1995**] M. A. Arbib, editor. *Handbook of Brain Theory and Neural Networks.* MIT Press, 1995.

[**Asimov, 1985**] Isaac Asimov. *Nueva guía de la Ciencia.* Plaza & Janés, 1985.

[**Axelrod, 1984**] Robert Axelrod. *The Evolution of Cooperation.* Basic Books, 1984.

[**Bacchus and Grove, 1996**] F. Bacchus and A. J. Grove. Utility independence in a qualitative decision theory. In *KR-96, Proc. of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 542–552. Morgan Kaufmann, 1996.

[**Bacchus, 1990**] Fahiem Bacchus. *Representing and Reasoning With Probabilistic Knowledge. A Logical Approach to Probabilities.* The MIT Press, 1990.

[**Barber, 2001**] Federico Barber, editor. *Inteligencia Artificial. Monografía: Desarrollo de Sistemas Multi-Agentes. Revista Iberoamericana de Inteligencia Artificial*, volume 13. AEPIA (Asociación Española para la Inteligencia Artificial, Verano 2001.

[**Barceinas, 1998**] Adolfo Barceinas. Un marco de comunicación interagentes en una biblioteca digital. Master's thesis, Universidad de las Américas-Puebla, 1998.

[**Barwise and Etchemndy, 2000**] Jon Barwise and John Etchemndy. *Language, Proof and Logic.* CSLI Publications / Seven Bridges Press, 2000.

[**Bates, 1994**] Joseph Bates. The role of emotion in believable agents. Technical Report CMU-CS-94-136, School of Computer Science, Carnegie Mellon University, April 1994.

[**Bender, 1996**] E.A. Bender. *Mathematical Methods in Artificial Intelligence.* IEEE Computer Society, 1996.

[**Bezdek, 1993**] Jim Bezdek. Fuzzy models - what are they, and why ? *IEEE Transactions on Fuzzy Systems*, 1(1), Febrero 1993.

[**Bigus and Bigus, 2001**] Joseph P. Bigus and Jennifer Bigus. *Constructing Intelligent Agents Using Java.* John Wiley and Sons, Inc., 2001.

[**Bishop, 1999**] Judy Bishop. *Java. Fundamentos de Programación.* Addison-Wesley, segunda edition, 1999.

[**Bond and Gasser, 1980**] A. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, 1980.

[**Brachman, 1979**] Ronald J. Brachman. *On the epistemological status of semantic networks*, pages 3–50. In Findler [**1979**], 1979.

[**Bratko, 1990**] Ivan Bratko. *Prolog. Programming for Artificial Intelligence.* Addison-Wesley, 1990.

[**Bratman, 1987**] Michael Bratman. *Intentions, Plans and Practical Reason.* Harvard University Press, 1987.

[**Brenner *et al.*, 1998**] Walter Brenner, Rüdiger Zarnekow, and Hartmut Wittig. *Intelligent Software Agents. Foundations and Applications.* Springer, 1998.

[**Brewka *et al.*, 1997**] Gerhard Brewka, Jürgen Dix, and Kurt Konolige. *Nonmonotonic Reasoning. An Overview*, volume 73 of *CSLI Lecture Notes.* CSLI Publications, Stanford, California, 1997.

[**Brewka, 1996**] Gerhard Brewka, editor. *Principles of Knowledge Representation.* CSLI Stanford, 1996.

[**Brooks, 1991a**] Rodney A. Brooks. *Elephants Don't Play Chess*, pages 3–15. In Maes [**1991b**], 1991.

[**Brooks, 1991b**] Rodney A. Brooks. Intelligence without reason. A.I. Memo 1239, MIT, Artifical Intelligence Laboratory, April 1991.

[**Brooks, 1991c**] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[**Buchanan and Shortliffe, 1984**] B. G. Buchanan and E. H. Shortliffe. *Ruled-Based Expert Systems.* Addison-Wesley, 1984.

[**Bussmann and Muller, 1992**] S. Bussmann and J. Muller. A negotiation framework for co-operating agents. In Dean [**1992**], pages 1–17.

[**Campoy, 2001**] Laura Campoy. *Esquema Formal para Integración de Componentes Reutilizables y Compartibles de Conocimiento en la Gestión de Memorias Corporativas.* PhD thesis, Dpto. de Ingeniería de la Información y las Comunicaciones. Universidad de Murcia, Octubre 2001.

[**Castel and Llorens, 1999**] Mª Jesús Castel and Faraón Llorens. *Lógica de Primer Orden.* Dpto. Ciencia de la Computació e Inteligencia Artificial. Universidad de Alicante, 1999.

[**Castillo *et al.*, 1996**] Enrique Castillo, José M. Gutiérrez, and Ali S. Hadi. *Sistemas Expertos y Modelos de Redes Probabilísticas.* Academia de Ingeniería, 1996.

[**Charniak and McDermott, 1985**] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, 1985.

[**Chauhan, 1997**] Deepika Chauhan. JAFMAS: A java-based agent framework for multiagent systems development and implementation. Master's thesis, ECECS Department, University of Cincinnati, 1997. http://www.ececs.uc.edu/ãbaker/JAFMAS.

[**Clarke, 1985**] Arthur C. Clarke. *2001. Una odisea espacial.* Ediciones Orbis, 1985.

[**Clips, 1993**] Clips. *CLIPS Version 6.0. User's Guide.* NASA, Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch, May 1993. http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/expert/systems/clips.

[**Clocksin and Mellish, 1987**] W. F. Clocksin and C. S. Mellish. *Programación en Prolog.* Gustavo Gili, 1987.

[**Cousins, 1994**] Steve B. Cousins. Intelligent interface agents. May 1994.

[**Craik, 1943**] K. J. Craik. *The Nature of Explanation.* Cambridge University Press, 1943.

[**Crevier, 1996**] Daniel Crevier. *Inteligencia Artificial.* Acento Editorial, 1996.

[**Crick, 1994**] Francis Crick. *La búsqueda científica del alma. Una revolucionaria hipótesis para el siglo XXI.* Circulo de lectores, 1994.

[**Crook, **] Crook. The nature of conscious awareness. *Mindwaves*, page 392.

[**Cuena, 1996**] José Cuena. *Notas sobre Modelos de Razonamiento.* Facultad de Informática. Universidad Polit'ecnica de Madrid, 1996.

[**Davis and Lenat, 1982**] R. Davis and D. B. Lenat. *Knowledge-Based Expert Systems.* McGraw-Hill, 1982.

[**Davis and Smith, 1983**] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, 1983.

[**Davis *et al.*, 1993**] Randall Davis, Howard Schrobe, and Peter Szolovits. What is a knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

[**Davis, 1970**] Morton D. Davis. *Game Theory: A Nontechnical Introduction.* Basic Books, 1970.

[**Davis, 1990**] E. Davis. *Representations of Commonsense Knowledge.* Morgan Kaufmann, 1990.

[**Dean, 1992**] S. M. Dean, editor. *Proc. CKBS-SIG.* Dake Centre, University of Keele, 1992.

[**Deitel and Deitel, 1998**] H. M. Deitel and P. J. Deitel. *Cómo programar en Java.* Prentice Hall, 1998.

[**Dempster, 1967**] A. P. Dempster. Upper and lower probabilities induced by a multivalued mapping. *Annals of Mathematical Statistics*, 38:325–339, 1967.

[**Dennett, 1994**] Daniel C. Dennett. Cognitive science as reverse engineering: Several meanings of "top-down" and "bottom-up". In D. Prawitz, B. Skyrms, and D. Westertahl, editors, *Proceedings of the 9th International Congress of Logic, Methodology and Philosophy of Science.* Amsterdam, North-Holland, 1994.

[**Dennett, 1999**] Daniel C. Dennett. *La peligrosa idea de Darwin.* Circulo de Lectores, 1999.

[**Doyle, 1979**] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[**Dubois and Prade, 1986**] D. Dubois and H. Prade. *Fuzzy Sets and Statistical Data.* European J. Oper. Res, 1986.

[**Duda *et al.*, 1977**] R. Duda, P. E. Hart, N. J. Nilsson, R. Reboh, J. Slocum, and G. Sutherland. Development of a computer-based consultant for mineral exporation. Technical report, SRI (Stanford Research Institute), 1977.

[**Durfee, 1988**] E. H. Durfee. *Coordination of Distributed Problem Solvers.* Kluwer Academic, 1988.

[**Durkin, 1994**] John Durkin. *Expert Systems. Design and development.* Prentice Hall, 1994.

[**Dym and Levitt, 1991**] C. Dym and R. Levitt. *Knowledge-Based Systems in Engineering.* McGraw-Hill, 1991.

[**Etzioni *et al.*, 1993**] Oren Etzioni, Henry M. Levy, Richard B. Segal, and Chandramohan A. Thekkath. OS Agents: Using AI Techniques in the Operating System Environment. Technical Report 93-04-04, Department of Computer Science and Engineering, University of Washington, April 1993.

[**Fagin *et al.*, 1996**] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vondi. *Reasoning about Knowledge.* The MIT Press, 1996.

[**Feigenbaum *et al.*, 1988**] E. Feigenbaum, P. McCorduch, and H. P. Nii. *The Rise of the Expert Company: How Visionary Companies Are Using Artificial Intelligence to Achieve Higher Productivity and Profits.* Times Books, 1988.

[**Findler, 1979**] Nicholas V. Findler, editor. *Associative Networks: Representation and Use of Knowledge by Computers.* Academic Press, 1979.

[**Finin *et al.*, 1994**] Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. *KQML: An Information and Knowledge Exchange Protocol.* In Fuchi and Yokoi [**1994**], 1994.

[**Forgy, 1982**] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[**Forner, 1993**] L. Forner. What's an agent, anyway? a sociological case study. Technical Report Agents Memo 93-01, MIT Media Lab, 1993.

[**Franklin and Graesser, 1996**] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proc. of the Third International Workshop on Agent Theories, Architectures, and Languages.* Springer-Verlag, 1996.

[**Franklin, 1995**] Stan Franklin. *Artificial Minds.* MIT Press, 1995.

[**Friedman-Hill, 2001**] Ernest J. Friedman-Hill. Jess, The Java Expert System Shell. Technical Report SAND98-8206 (revised), Distributed Computing Systems, Sandia National Laboratories, September 2001.

[**Fuchi and Yokoi, 1994**] Kazuhiro Fuchi and Toshio Yokoi, editors. *Knowledge Building and Knowledge Sharing.* Ohmsha and IOS Press, 1994.

[**FuzzyCLIPS, 1994**] FuzzyCLIPS. *FuzzyCLIPS Version 6.02A. User's Guide.* Knowledge Systems Laboratory, Institute for Information Technology, National Research Council Canada, September 1994.

[**Gabbay *et al.*, 1994**] Dov. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors. *Nonmonotonic Reasoning and Uncertain Reasoning*, volume 3 of *Handbook of Logic in Artificial Intelligence and Logic Programming.* Oxford University Press, 1994.

[**Gardner, 1998**] Howard Gardner. *Inteligencias Múltiples.* Editorial Paidós, 1998.

[**Gelder, 1992**] T. J. van Gelder. Defining "distributed representation". *Connection Science*, 4:175–191, 1992.

[**Genesereth and Fikes, 1992**] Michael R. Genesereth and Richard E. Fikes. *Knowledge Interchange Format, version 3.0 Reference Manual.* Stanford University Logic Group, 1992. http://www-ksl.stanford.edu/knowledge-sharing/kif.

[**Genesereth and Fikes, 1998**] Michael R. Genesereth and Richard E. Fikes. *Knowledge Interchange Format (KIF).* draft proposed American National Standard NCITS.T2/98-004, 1998. http://logic.stanford.edu/kif/dpans.html.

[**Genesereth and Ketchpel, 1994**] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

[**Genesereth and Nilsson, 1987**] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers, 1987.

[**Giannesini *et al.*, 1989**] F. Giannesini, H. Kanoui, R. Pasero, and M. van Caneghem. *Prolog.* Addison-Wesley Iberoamericana, 1989.

[**Giarratano and Riley, 1994**] Joseph Giarratano and Gary Riley. *Expert Systems: Principles and Programming.* PWS Publishing, 2nd edition, 1994.

[**Giarratano, 1998**] Joseph C. Giarratano. *CLIPS User's Guide. Version 6.10*, August 1998. http://www.ghg.net/clips/CLIPS.html.

[**Gmytrasiewicz *et al.*, 1991**] P. J. Gmytrasiewicz, E. H. Durfee, and D. K. Wehe. A decision-theoretic approach to cordinating multiagent interactions. In *Proc. of the Twelfth International Joint Conference on Artificial Intelligence*, pages 62–68, 1991.

[**Goleman, 1998**] Daniel Goleman. *Inteligencia Emocional.* Editorial Kairós, 1998.

[**Gómez-Pérez and Benjamins, 1999**] A. Gómez-Pérez and V. R. Benjamins. Overview of knowledge sharing and reuse components: Ontologies and problem-solving methods. In *Proc. of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5).* Stockholm, Sweden, 1999.

[**González, 1998**] Mª José González. *Introducción a la Psicología del Pensamiento.* Cognitiva. Trotta, 1998.

[**Green *et al.*, 1997**] S. Green, L. Hurst, B.Ñangle, P. Cunningham, F. Somers, and R. Evans. Software agents: A review. Technical report, Trinity College Dublin y Broadcom Éireann Research Ltd., May 1997. http://www.cs.tcd.ie/Brenda.Nangle/iag.html.

[**Gruber, 1991**] T. Gruber. Ontolingua: A mechanism to support portable ontologies. Technical Report KSL-91-66, Stanford Knowledge Systems Laboratory, 1991.

[**Guha and Lenat, 1990**] R. V. Guha and D. Lenat. Cyc: A midterm report. *AI Magazine*, pages 33–59, Fall 1990.

[**Guilfoyle, 1995**] C. Guilfoyle. Vendors of agent technology. UNICOM Seminar on Intelligent Agents and their Business Applications, November 1995.

[**Haddadi, 1996**] A. Haddadi. *Communication and cooperation in agent systems. A pragmatic theory*, volume 1056 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 1996.

[**Haddawy and Hanks, 1992**] P. Haddawy and S. Hanks. Rpresentations for decision-theoretic planning: Utility functions for deadline goals. In *KR-92, Proc. of the Third International Conference on the Principles of Knowledge Representation and Reasoning*, pages 71–82. Morgan Kaufmann, 1992.

[**Halpern, 1990**] Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.

[**Hayes-Roth, 1985**] B. Hayes-Roth. A blackboard arquitectures for control. *Artificial Intelligence*, 26(3):251–321, 1985.

[**Hilera and Martínez, 1995**] José R. Hilera and Víctor J. Martínez. *Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones.* Ra-ma, 1995.

[**Hofstadter, 1995**] Douglas R. Hofstadter. *Fluid Concepts and Creative Analogies. Computer Models of the Fundamental Mechanisms of Thought.* Penguin Books, 1995.

[**Hofstadter, 1998**] Douglas R. Hofstadter. *Gödel, Escher, Bach. Un Eterno y Grácil Bucle*, volume 14 of *Metatemas.* Tusquets Editores, sexta edition, 1998.

[**Humphrey, 1995**] Nicholas Humphrey. *Una Historia de la Mente.* Ciencias Cognitivas. Editorial Gedisa, 1995.

[**Jagannathan *et al.*, 1989**] V. Jagannathan, R. Dodhiawala, and L. S. Baum, editors. *Blackboard Architectures and Applications.* Academic Press, 1989.

[**Kahneman *et al.*, 1982**] D. Kahneman, P. Slovic, and A. Tversky, editors. *Judgment under Uncertainty: Heuristics and Biases.* Cambridge University Press, 1982.

[**Khalfa, 1996**] Jean Khalfa. *What Is Intelligence?* Ciencias Cognitivas. Cambridge University Press, 1996.

[**King, 1995**] J. A. King. Intelligent agents: Bringing good things to life. *AI Expert*, pages 17–19, February 1995.

[**Klir and Yuan, 1995**] George J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic. Theory and Applications.* Prentice Hall, 1995.

[**Kosko, 1995**] Bart Kosko. *Pensamiento Borroso. La nueva ciencia de la lógica borrosa.* Crítica, Grijabo Mondadori, 1995.

[**Kosko, 2000**] Bart Kosko. *El futuro borroso o el mundo en un chip.* Crítica, Grijabo Mondadori, 2000.

[**Kowalski, 1986**] Robert Kowalski. *Lógica, programación e inteligencia artificial.* Díaz de Santos, 1986.

[**Kumar and Feldman, 1998**] M. Kumar and S. I. Feldman. Business negotiation on the internet. Technical report, IBM Institute for Advanced Commerce (IAC), 1998. http://www.ibm.com/iac/reports-technical/reports-bus-neg-internet.html.

[**Kuokka and Harada, 1995**] D. Kuokka and L. Harada. On using KQML for matchmaking. In *Proc. of the First International Conference on Multiagent Systems*, 1995.

[**Labrou and Finin, 1997**] Yannis Labrou and Tim Finin. A proposal for a new KQML specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, February 1997.

[**Langton, 1989**] Christopher Langton, editor. *Artificial Life.* Addison-Wesley, 1989.

[**Lenat and Guha, 1990**] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems.* Addison-Wesley, 1990.

[**Llorens and Mira, 2000**] Faraón Llorens and Sergio Mira. ADN (Asistente para Deducción Natural) Natural Deduction Assistant. In *First International Congress on Tools for Teaching Logic.* Salamanca, June 2000.

[**Lloyd, 1993**] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, second extended edition, 1993.

[**Luce and Raiffa, 1957**] R. Luce and H. Raiffa. *Games and Decisions.* John Wiley & Sons, 1957.

[**Luger and Stubblefield, 1999**] George F. Luger and William A. Stubblefield. *Artificial Intelligence. Structures and Strategies for Complex Problem Solving.* Addison-Wesley, third edition, 1999.

[**Luger, 1995**] George F. Luger, editor. *Computation & Intelligence. Collected Readings.* AAAI Press / The MIT Press, 1995.

[**Maes, 1991a**] P. Maes. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, chapter Situated Agents Can Have Goals, pages 49–70. In [**1991b**], 1991.

[**Maes, 1991b**] P. Maes, editor. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back.* MIT Press, 1991.

[**Maes, 1995**] P. Maes. Intelligent software. *Scientific American*, 273(3), September 1995.

[**Marcel and Bisiach, 1988**] A. J. Marcel and E. Bisiach, editors. *Consciousness in Contemporary Science.* Clarendon Press, 1988.

[**Marcel, 1988**] Anthony J. Marcel. *Phenomenal Experience and Functionalism*, page 121. In Marcel and Bisiach [**1988**], 1988.

[**Maynard-Smith, 1982**] John Maynard-Smith. *Evolution and the Theory of Games.* Cambridge University Press, 1982.

[**McCarthy and Hayes, 1969**] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Learning*, 4, 1969.

[**McCarthy and Lifschitz, 1990**] J. McCarthy and V. Lifschitz, editors. *Formalizing Common Sense.* Ablex, 1990.

[**McCarthy, 1959**] John McCarthy. Programs with common sense. http://www-formal.stanford.edu/jmc/mcc59/mcc59.html, 1959.

[**McCarthy, 1980**] John McCarthy. Circumscription - A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 13:27-39, 1980.

[**McCulloch and Pitts, 1948**] W. McCulloch and W. Pitts. The statistical organization of nervous activity. *Journal of the American Statistical Association*, 4:91-99, 1948.

[**McDermott and Doyle, 1980**] Drew McDermott and Jon Doyle. Non-Monotonic Logic I. *Artificial Intelligence*, 13:41-72, 1980.

[**McGinn, 1989**] Colin McGinn. Can we solve the mind-body problem? *Mind*, pages 349-366, 1989.

[**McNeill and Freiberger, 1993**] Daniel McNeill and Paul Freiberger. *Fuzzy Logic.* Touchstone, 1993.

[**Mendel, 1995**] Jerry M. Mendel. Fuzzy logic systems for engineering : A tutorial. IEEE, 1995.

[**Minsky, 1968**] M. Minsky, editor. *Semantic information processing.* M.I.T. Press, 1968.

[**Minsky, 1974**] Marvin Minsky. A framework for representing knowledge. Technical Report 306, MIT Artificial Intelligence Laboratory, 1974.

[**Minsky, 1985**] M. Minsky. *The Society of Mind.* Simon & Schuster, 1985.

[**Moore, 1985**] Robert C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25, 1985.

[**Morgan, 1999**] Mike Morgan. *Descubre Java 1.2.* Prentice Hall, 1999.

[**Neches *et al.*, 1991**] R.Ñeches, R. E. Fikes, T. Finin, T. R. Gruber, R. Patil, T. Senator, and W. R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36-56, 1991.

[**Nerode and Shore, 1997**] Anil Nerode and Richard A. Shore. *Logic for Applications.* Springer, second edition, 1997.

[**Neumann and Morgenstern, 1944**] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[**Newell and Simon, 1972**] A.Ñewell and H. A. Simon. *Human Problem Solving.* Prentice-Hall, 1972.

[**Nii, 1986a**] H. P. Nii. Blackboard systems (part two): Blackboard application systems, blackboard systems from a knowledge engineering perspective. *The AI Magazine*, 7(3):82–106, 1986.

[**Nii, 1986b**] H. P. Nii. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard arquitectures. *The AI Magazine*, 7(2):38–64, 1986.

[**Nilsson, 1986**] Nils J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–88, 1986.

[**Nilsson, 2001**] Nils J. Nilsson. *Inteligencia Artificial. Una nueva síntesis.* McGraw-Hill, 2001.

[**Nwana, 1996**] Hyacinth S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–244, October/November 1996.

[**Palma *et al.*, 2000**] J. T. Palma, E. Paniagua, F. Martín, and R. Marín. Ingeniería del Conocimiento. De la Extracción al Modelado de Conocimiento. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, 11:46–72, 2000.

[**Penrose, 1991**] Roger Penrose. *La Nueva Mente del Emperador.* Mondadori, 1991.

[**Penrose, 1999**] Roger Penrose. *Lo Grande, lo Pequeño y la Mente Humana.* Cambrige University Press, 1999.

[**Petrie, 1996**] Charles J. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, December 1996.

[**Pinker, 1998**] Steven Pinker. *How the Mind Works.* Allen Lane The Pinguin Press, 1998.

[**Poundstone, 1995**] William Poundstone. *El dilema del prisionero. John von Neumann, la teoría de juegos y la bomba.* Alianza Editorial, 1995.

[**Pujol *et al.*, 2001**] F. Pujol, M. Pujol, F. Llorens, R. Rizo, and J.M. García. Diseño de un sistema experto para el diagnostico de trastornos mentales. In *6to. Simposio de Matemática y IV Conferencia Italo-Latinoamericana de Matemática Aplicada e Industrial. CIMAF'2001 (Conferencia Internacional Ciencia y Tecnología para el Desarrollo).* La Habana, Cuba, Marzo 2001.

[**Quillian, 1968**] M. Ross Quillian. *Semantic information processing*, chapter Semantic Memory. M.I.T. Press, 1968.

[**Rao and Georgeff, 1995**] A. Rao and M. Georgeff. BDI Agents: From Theory to Practice. In *Proc. of the First International Conference on Multiagent Systems*, 1995.

[**Reeves and Clarke, 1990**] Steve Reeves and Michael Clarke. *Logic for Computer Science.* Addison-Wesley, 1990.

[**Reiter, 1980**] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[**Rich and Knight, 1994**] Elaine Rich and Kevin Knight. *Inteligencia Artificial.* McGraw-Hill, 1994.

[**Rizo, 2001**] Ramón Rizo. 2001: Componentes inteligentes artificiales. Lección Inaugural curso académico 2001-2002. Universidad de Alicante, septiembre 2001.

[**Romero, 2000**] Francisco Javier Romero. Sistema JAI: Sistema de multiagentes inteligentes basados en java. Master's thesis, Dpto. Ciencia de la Computación e Inteligencia Artificial. Escuela Politénica Superior, 2000.

[**Rosenschein and Genesereth, 1985**] J. S. Rosenschein and M. R. Genesereth. Deals among rational agents. In *Proc. of the Ninth International Joint Conference on Artificial Intelligence*, pages 91–99. AAAI Press, 1985.

[**Rosenschein and Zlotkin, 1994**] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers.* MIT Press, 1994.

[**Rosenschein, 1985**] J. S. Rosenschein. *Rational Interaction: Cooperation Among Intelligent Agents.* PhD thesis, Stanford University, 1985.

[**Russell and Norvig, 1996**] Stuart Russell and Peter Norvig. *Inteligencia Artificial. Un enfoque moderno.* Prentice Hall, 1996.

[**Schank and Abelson, 1977**] R. C. Schank and R. Abelson. *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum Associates, 1977.

[**Schreiber *et al.*, 1993**] G. Schreiber, B. Wielinha, and J. Breuker, editors. *KADS: A Principled Approach to Knowledge-Based System Development.* Academic Press, 1993.

[**Searle, 1969**] J. Searle. *Speech Acts: An Essay in the Philosofy of Language.* Cambridge University Press, 1969.

[**Shafer, 1976**] Glen Shafer. *A Mathematical Theory of Evidence.* Princenton University Press, 1976.

[**Shannon, 1948**] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, 1948.

[**Shoham, 1993**] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[**Shortliffe, 1976**] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* Elsevier, 1976.

[**Siekmann and Wrightson, 1983**] Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning 2. Classical Papers on Computational Logic 1967-1970.* Springer-Verlag, 1983.

[**Simon, 1996**] Herbert A. Simon. *The Sciences of the Artificial.* The MIT Press, third edition, 1996.

[**Sloman,**] Aaron Sloman. Synthetic minds. School of Computer Science, The University of Birmingham, http://www.cs.bham.ac.uk/ãxs/misc/synthetic.minds.html.

[**Smith and Davis, 1981**] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1), 1981.

[**Smith, 1980**] R. G. Smith. *The Contract Net Protocol: High-level communication and control in a distributed problem solver.* In Bond and Gasser [**1980**], 1980.

[**Socher-Ambrosius and Johann, 1997**] Rolf Socher-Ambrosius and Patricia Johann. *Deduction Systems.* Graduate texts in computer science. Springer, 1997.

[**Sowa, 2000**] John F. Sowa. *Knowledge Representation. Logical, Philosophical, and Computational Foundations.* Brooks/Cole, 2000.

[**Sun Microsystems, 2001**] Sun Microsystems, The Java Tutorial. A practical guide for programmers, 2001. http://java.sun.com/docs/books/tutorial.

[**Sycara, 1998**] K. Sycara. Multiagent system. *AI Magazine*, 19(2):79–92, 1998.

[**Thorpe, 1995**] S. Thorpe. *Handbook of Brain Theory and Neural Networks*, chapter Localized versus distributed representations, pages 549–552. In Arbib [**1995**], 1995.

[**Varela, 1998**] Pilar Varela. *La Máquina de Pensar.* Editorial Temas de Hoy, 1998.

[**JATLite web, 1998**] JATLite web, Java Agent Template, Lite, 1998. http://java.stanford.edu.

[**Aglets web, 2001**] Aglets web, Aglets Software Development Kit, 2001. http://www.trl.ibm.com/aglets.

[**CYC web, 2001**] CYC web, Proyecto CYC, 2001. http://www.cyc.com.

[**Jess web, 2001**] Jess web, Jess, The Expert System Shell for the Java Platform, 2001. http://herzberg.ca.sandia.gov/jess.

[**KIF web, 2001**] KIF web, Knowledge Interchange Format, 2001. http://www-ksl.stanford.edu/knowledge-sharing/kif.

[**KQML web, 2001**] KQML web, Knowledge Query and Manipulation Language Web, 2001. http://www.cs.umbc.edu/kqml.

[**KSE web, 2001**] KSE web, ARPA Knowledge Sharing Effort, 2001. http://www-ksl.stanford.edu/knowledge-sharing.

[**KSL web, 2001**] KSL web, Knowledge Systems Laboratory. Stanford University, 2001. http://www-ksl.stanford.edu.

[**RoboCup web, 2001**] RoboCup web, RoboCup soccer competition, 2001. http://www.robocup.org.

[**Weiss, 2000**] Mark Allen Weiss. *Estructuras de datos en Java.* Addison-Wesley, 2000.

[**Westerstahl, 2000**] Dag Westerstahl. Special issue on logics of uncertainty. *Journal of Logic, Language and Information*, 9(1), January 2000.

[**Wiener, 1998**] Norbert Wiener. *Cibernética o El control y comunicación en animales y máquinas.* Tusquets Editores, 2 edition, 1998. Primera edición en inglés de 1948.

[**Wilkes, 1988**] Kathleen V. Wilkes. *Consciousness in Contemporary Science*, chapter -, Yishi, Duh, Um, and Consciousness, page 38. In Marcel and Bisiach [**1988**], 1988.

[**Wilson and Keil, 1999**] Robert A. Wilson and Frank C. Keil, editors. *The MIT Enciclopedia of the Cognitive Sciences.* The MIT Press, 1999.

[**Winston, 1994**] Patrick H. Winston. *Inteligencia Artificial.* Addison-Wesley Iberoamericana, tercera edition, 1994.

[**Wooldridge and Jennings, 1995**] Michael J. Wooldridge and Nicholas R. Jennings, editors. *Intelligent Agents*, volume 890 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1995.

[**Wooldridge *et al.*, 1996**] M. J. Wooldridge, J. P. Mueller, and M. Tambe, editors. *Intelligent Agents II*, volume 1037 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1996.

[**Zadeh, 1965**] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[**Zlotkin, 1994**] G. Zlotkin. *Mechanisms for Automated Negotiation among Autonomous Agents*. PhD thesis, Hebrew University, February 1994.

APÉNDICES

Apéndice A

NATURALEZA DE LA INTELIGENCIA

> El grado de inteligencia que atribuimos al comportamiento de algo está determinado tanto por nuestro propio estado mental y nuestra capacitación como por las propiedades del objeto que analizamos
>
> Alan Turing

Definir la inteligencia no es fácil. La inteligencia está siendo estudiada desde distintas disciplinas: neurociencia, ciencia cognitiva, ciencia de la computación, lógica, psicología, lingüística, antropología, etc. Y por ello, hablar de inteligencia implica manejar muchos conceptos y moverse con cierta soltura en distintos campos científicos. El estudio de la mente humana está en auge en campos tan dispares como la publicidad, la política o la inteligencia artificial (ciencia de la computación). La intención al incluir este capítulo es que sirva de introducción al campo de la Inteligencia Artificial y que se tenga una visión general de la inteligencia más allá de los procedimientos y técnicas computacionales; en ningún momento se ha pretendido hacer un tratado de lo que es la inteligencia.

Una primera ventaja al estudiar la inteligencia es que todos somos, en parte, psicólogos y conocemos lo que es la mente humana ya que vivimos con ella. Pero esto representa al mismo tiempo una gran desventaja, ya que al formar parte de nosotros no podemos estudiarla desde fuera de manera objetiva. Es difícil definir cómo trabaja la mente humana porque "pensar no requiere ningún libro de instrucciones" ([**Varela, 1998**]). Parece fácil pero es muy complejo. Aún hay mucho por descubrir y está rodeado de misterio.

El verdadero problema no es si las máquinas piensan, sino si lo hacen los hombres

Muchos son los interrogantes que se nos plantean al empezar a hablar de la inteligencia: ¿qué es ser inteligente?, ¿es lo mismo ser listo que ser

inteligente?, ¿puede medirse la inteligencia? y, la pregunta clave desde nuestro campo de acción, ¿puede una máquina ser inteligente? Las respuestas no son sencillas y no pretendemos responderlas aquí. Aunque una conclusión que queremos dejar clara es que hay muchos modos de ser inteligente.

A.1 Enfoque biológico de la Inteligencia

A.1.1 Biología como Ingeniería

En este apartado vamos a tratar aquellos aspectos de la biología que nos interesen, pero vistos desde la perspectiva de la ingeniería. El matrimonio entre la biología y la ingeniería es el hecho central de la revolución darwiniana. La idea del estudio de los seres vivos de forma cercana a la ingeniería es antigua, pero es en los últimos tiempos cuando se empieza a reconocer que la ingeniería recoge en su seno algunos pensamientos importantes [**Simon, 1996**]. La barrera que separa lo *natural* de lo *artificial* puede desmoronarse si existe un espacio de diseño en el que nuestros cuerpos y nuestras mentes se encuentran unidos en un continuo proceso de I+D (Investigación y Desarrollo). La gran visión de Darwin fue que todos los diseños de la biosfera podrían ser los productos de un proceso *automático* de gradual *elevación* en el espacio del diseño. Una detallada reflexión sobre el significado de la teoría darwiniana en la visión actual del mundo la podemos encontrar en [**Dennett, 1999**]. Pero en cuanto al tema que nos ocupa, es decir la relación de la inteligencia con la evolución, en *2001. Una odisea en el espacial*, obra de ciencia ficción que trata el tema de la inteligencia, se dice "al no estar ya semiembotados por la inanición, disponían de tiempo para el ocio y para los primeros rudimentos de pensamiento" ([**Clarke, 1985**]). Pero el proceso de elevación en el espacio del diseño, tal como aparece en [**Penrose, 1999**], vendría dado en que "para nuestros ancestros remotos, una capacidad específica para hacer matemáticas sofisticadas difícilmente puede haberles sido una ventaja selectiva, pero sí pudo haberlo sido perfectamente una capacidad general para comprender".

La mente es un sistema de computación diseñado por selección natural

La estrategia de interpretar los organismos como si fueran artefactos y ponernos a estudiarlos lo podemos considerar como proceso de *reverse-engineering* (ingeniería inversa o revertida) [**Dennett, 1994**]. En la ingeniería se diseñan máquinas para hacer algo. En la ingeniería inversa nos imaginamos para hacer qué cosa fue diseñada la máquina. Así, cuando una compañía quiere construir un aparato electrónico para competir con el de otra compañía, compra un aparato de la competencia y procede a analizarlo: se lleva a un banco de pruebas donde se hace

funcionar, se desmenuza y se analiza cada una de sus partes. En este momento nos hacemos un montón de preguntas (*¿por qué?*) y buscamos las respuestas: "todo tiene su razón de ser". Nos ponemos en una *posición intencional* tratando de deducir lo que los diseñadores tenían en mente cuando hicieron su trabajo. Se trata al artefacto sometido a examen como el producto de un proceso de desarrollo razonado de un diseño, a través de una serie de elecciones entre diversas alternativas, en las cuales las decisiones tomadas son aquellas que los diseñadores consideraron mejores.

No deberemos ser demasiado estrictos al tratar el carácter óptimo de todos los aspectos, ya que a veces los ingenieros ponen cosas inútiles en sus diseños, se les olvida quitar cosas que ya no cumplen una función o no se dan cuenta de posibles mejoras en el diseño. Puede incluso que alguna de las características finales no desempeñe ninguna función en el producto acabado, aunque haya tenido un papel crucial durante el proceso de construcción. Debemos tener presente que cualquier organismo o artefacto ha llegado a su forma actual mediante un proceso que tiene sus propios requerimientos y, por tanto, para que funcione en cada etapa del proceso se tendrán que cumplir rígidas restricciones que limitarán sus posteriores características.

A.1.2 Mente y Cerebro

La mente humana es la mejor obra de arte de la naturaleza

En los últimos años, gracias a la evolución de la Neurociencia, y a la ayuda de ordenadores y escáneres, la química molecular y otros avances científicos y técnicos, conocemos mejor el cerebro humano y su funcionamiento. El cerebro es la parte más delicada de nuestro cuerpo, que con apenas kilo y medio de peso, contiene alrededor de 30.000 millones de neuronas y 100 trillones de interconexiones, tanto en serie como en paralelo. En su interior se encuentran nuestros pensamientos y recuerdos, nuestras emociones y pensamientos.

Las neuronas son las unidades funcionales y anatómicas del cerebro. Constan de un cuerpo celular o soma del que parten numerosas ramificaciones llamadas dendritas y una ramificación principal o axón. Cada célula nerviosa está conectada con otras neuronas por medio de las dendritas que reciben información procedente de otras neuronas y el axón que conduce la respuesta hacia otras células nerviosas. Las dendritas procedentes de otras neuronas se comunican con el cuerpo celular a través de unas zonas conocidas como sinapsis. La transmisión de señales a través de estos espacios sinápticos se realiza mediante sustancias químicas o neurotransmisores.

Cuando nacemos el cerebro es como un libro en blanco que con los es-

tímulos recogidos por nuestros sentidos va organizando sus neuronas y configurando sus conexiones. De esta forma pasamos nuestros primeros años de vida conectando neuronas.

La evolución ha hecho aumentar el tamaño del cerebro. "Nuestros cerebros altamente desarrollados, después de todo, no han evolucionado a partir de la presión del descubrimiento de verdades científicas sino solamente para permitirnos ser suficientemente listos como para sobrevivir y dejar descendencia" ([**Crick, 1994**]). Pero no sólo hay cambios cuantitativos, también ha dado saltos cualitativos. Tal como dice Stephen Hawking "la evolución desde los cerebros de lombrices hasta los cerebros humanos tuvo lugar presumiblemente por selección natural darwiniana. La cualidad seleccionada era la capacidad para escapar de los enemigos y para reproducirse, y no la capacidad para hacer matemáticas. Por eso, una vez más, el teorema de Gödel no es relevante. Se trata simplemente de que la inteligencia necesaria para la supervivencia puede utilizarse también para construir demostraciones matemáticas" ([**Penrose, 1999**]).

Si el cerebro del ser humano fuera tan sencillo que lo pudiéramos entender, entonces seríamos tan estúpidos que tampoco lo entenderíamos

Pero un tema clave al hablar de la mente es el de la *conciencia.* Por la delicadeza del tema voy a trascribir algunas afirmaciones sobre la misma ([**Humphrey, 1995**]):

Conciencia es la máxima invención en la historia de la vida; ha permitido que la vida se dé cuenta de sí misma (Stephen Jay Gould).

La percatación consciente es una propiedad condicional del modelo de realidad en su forma tripartita. Puede decirse que es el aspecto subjetivo de la continua representación de un despliegue informal temporalmente estabilizado dentro del cual puede tener lugar el procesamiento multilateral de un asunto (John Crook).

En todos los contextos en los cuales tienden a ser desplegados, los términos "consciente" y afines son, para los efectos *científicos*, tan inútiles como innecesarios (Kalhleen Wilkes).

La referencia a la conciencia en la ciencia psicológica es requerida, legítima y necesaria. Es requerida ya que la conciencia es un (si no *el*) aspecto central de la vida mental. Es legítima porque hay fundamentos tan razonables para identificar la conciencia como los hay para identificar otros constructos psicológicos. Es necesaria porque posee valor explicativo, y en tanto que hay fundamentos para postular que tiene categoría causal (Anthony Marcel).

> Encuentro que no poseo un concepto claro de a qué se refiere la gente cuando habla acerca de la "conciencia" o de la "percatación fenomenal" (Alan Allport).
>
> Sentimos que, de algún modo, el agua del cerebro físico se convierte en el vino de la conciencia, pero estamos por completo in albis acerca de la naturaleza de esta conversión /.../ El *problema mente-cuerpo* es el problema de entender cómo se produce el milagro (Colin McGinn).

Citas sacadas de **[Crook,]**, **[Wilkes, 1988]**, **[Marcel, 1988]**, **[Allport, 1988]** y **[McGinn, 1989]**.

Un tratamiento más extenso de cómo trabaja la mente se puede encontrar en **[Pinker, 1998]**, y una visión rápida en **[Asimov, 1985]**

A.2 La Inteligencia

La primera dificultad con que nos encontramos al querer estudiar la inteligencia es su propia definición. No es sencillo definir la inteligencia, incluso los especialistas del tema no se ponen de acuerdo. Es más, el concepto de inteligencia está cambiando en los últimos tiempos. Veamos primero qué entendemos por inteligencia, para pasar posteriormente a analizar nuevas teorías sobre la inteligencia (inteligencias múltiples e inteligencia emocional) así como el origen de la inteligencia (genético o aprendido).

A.2.1 ¿Qué es la inteligencia?

Pese a que no existe un acuerdo unánime en su definición, si que se está de acuerdo en una serie de rasgos generales (**[Khalfa, 1996]**):

- La inteligencia es una capacidad que, en mayor o menor medida, todos poseemos.
- No es patrimonio exclusivo de los genios.
- No hay una única inteligencia.

Tradicionalmente, el mundo occidental ha puesto un acento excesivo en la "cultura", y su determinación de personas inteligentes está muy condicionado por este sesgo. Se habla de personas inteligentes de acuerdo a nuestros parámetros. Pero no debemos confundir ser inteligente con ser "culto", porque ¿es más inteligente un ingeniero que un labrador? Realmente ¿sabrá adaptarse mejor a su medio, llevar una mejor vida o ser más feliz un ingeniero que un labrador? La respuesta no es fácil ni tajante. Lo que si que parece claro es que no todas las personas tienen la misma capacidad intelectual. En palabras de Marvin Minsky, "en todo caso, ¿qué es la inteligencia? Es sólo una palabra con que nos referimos a los procesos desconocidos con los cuales nuestro cerebro resuelve problemas que consideramos difíciles. Sin embargo, cuando logramos dominar una habilidad, ya no nos impresiona ni mistificamos el hecho de que otras personas hagan la misma tarea. es por ello que el significado de *inteligencia* nos parece tan escurridizo: no describe algo definido, sino únicamente el horizonte temporal de nuestra ignorancia en cuanto a cómo puede funcionar la mente." ([**Minsky, 1985**]).

Lo importante no es tener mucha inteligencia sino saber aprovechar la que se tiene

Algunos psicólogos denominan *inteligencia general* o *Factor G* a esa capacidad común a todas las personas inteligentes y que describen como la habilidad para razonar y pensar en abstracto. Son lo que solemos llamar personas "listas". Se trataría de esa capacidad de previsión, de anticiparse a los acontecimientos, de analizar la situación a fin de adaptarse con éxito a las circunstancias, e independiente de la instrucción recibida.

Otros psicólogos consideran que las cualidades mentales de una persona son demasiado diversas para estar englobadas en un solo Factor G, y describen la inteligencia como una composición de distintas habilidades mentales. Unas personas están más capacitadas para realizar unas tareas que para otras; y a estas habilidades distintas las llaman *aptitudes.* Estas aptitudes nos determinan para elegir unos estudios u otros, unos trabajos determinados y hablamos de que hacemos mejor determinadas cosas. Así, L.L. Thurstone, en lo que denominó *inteligencia factorial* identificó siete factores que llamó *habilidades mentales primarias.* Entre estas habilidades primarias se encuentran la *inteligencia verbal* que nos permite expresarnos, tanto de forma oral como escrita, de modo correcto y utilizando los conceptos adecuados; la *inteligencia numérica* determina nuestra capacidad para realizar cálculos correctos con rapidez; la *inteligencia espacial* nos permite entender el mundo físico, sus espacios y sus volúmenes.

Como veremos en el siguiente apartado, H. Gardner, para recoger la diversidad del ser inteligente, considera que existen siete clases de inteligencia, que todos poseemos en cierto grado, y para ello utiliza el concepto de inteligencia múltiple.

Hoy en día se habla de *inteligencia social*, que son las habilidades que nos permiten triunfar en la vida, tanto en el ámbito profesional como el personal. No siempre los más "listos", los que obtienen mejores calificaciones en las escuela, son los que tienen éxito en la vida. Esta inteligencia tiene una diferencia más marcada con las demás inteligencias claramente académicas. Robert Sternberg, profesor de la Universidad de Yale, está investigando sobre lo que él llama *inteligencia práctica*, que debemos utilizar al organizar las tareas tanto en el trabajo como en la vida cotidiana. Esta inteligencia justifica el porqué personas con poca formación académica consiguen gran éxito social. En un apartado posterior trataremos más detenidamente lo que Daniel Goleman llama *inteligencia emocional*.

Para desenvolverse en el mundo con éxito y felicidad no es preciso ser un genio

Otros trabajos muy interesantes nos hablan de dos tipos complementarios de inteligencia, la *inteligencia cristalizada*, que es el resultado de lo que sabemos, de la experiencia, y la *inteligencia fluida*, que es aquella que nos permite abstraer y analizar diferencias entre conceptos. Así, mientras la inteligencia fluida se iría perdiendo conforme envejece el cuerpo, la inteligencia cristalizada aumentaría con la edad. De esta forma, las personas con la edad utilizan un mecanismo compensatorio, y a medida que pierde inteligencia fluida gana en inteligencia cristalizada, con lo que elude aquellas tareas en las que está en peores condiciones y opta por situaciones que domina.

Para finalizar este apartado un breve comentario. Como se ha visto, no existe "una" inteligencia sino "muchas", y todas ellas son útiles y necesarias.

A.2.2 Las Inteligencias Múltiples

Howard Gardner, profesor de la Universidad de Harvard, considera que la inteligencia no es algo que está dentro del individuo, sino que es el resultado de la interacción de la persona con el ambiente que la rodea ([**Gardner, 1998**]). Considera que no se puede preguntar si una persona es inteligente sino en qué se es inteligente. En su búsqueda de la verdadera esencia de la inteligencia, propone siete clases de inteligencia independientes, *Inteligencia Múltiple*, que todas las personas poseen en mayor o menor grado:

inteligencia lingüística : habilidad para utilizar el lenguaje. La poseen en alto grado los periodistas, escritores, políticos, ... Un representante de este tipo de inteligencia sería Shakespeare.

inteligencia lógico-matemática : habilidad para realizar cálculos numéricos. La poseen en alto grado los inge-

nieros, ... Representada por Einstein.

inteligencia espacial : habilidad para interpretar y entender el mundo físico. La poseen en alto grado los arquitectos, pintores, diseñadores, ... Un ejemplo sería Picasso.

inteligencia cinestésica : habilidad para utilizar el cuerpo o partes del mismo para resolver problemas. La poseen en alto grado los deportistas, cirujanos, artesanos, acróbatas del circo, ... Por ejemplo Nureyev.

inteligencia musical : La poseen en alto grado los cantantes, compositores, músicos, ... Mozart sería un notorio representante.

inteligencia interpersonal : capacidad de comprender a los demás, de establecer buenas relaciones con ellos. Los médicos, profesores, actores, vendedores, ... suelen ser individuos con alto nivel de este tipo de inteligencia. Gandhi poseía un alto nivel de inteligencia interpersonal.

inteligencia intrapersonal : capacidad para entenderse a sí mismo, que nos permite configurar una imagen exacta y verdadera de nosotros mismos y nos hace ser capaces de utilizar esa imagen para actuar de un modo más eficaz en la vida. Un exponente de este tipo de inteligencia sería Proust.

A.2.3 La Inteligencia Emocional

La mayoría de los sistemas educativos (tanto en los niveles primarios, secundarios como universitarios) se basan fundamentalmente en los aspectos matemáticos, verbales, lógicos y espaciales. Hemos reducido, peligrosamente, la inteligencia a su dimensión cognoscitiva, pensando que la inteligencia sólo servía para resolver problemas técnicos. Por ello en la sociedad actual hay una fuerte carencia de la habilidad para resolver problemas afectivos. Quién es más inteligente, ¿aquella persona que resuelve ecuaciones diferenciales o aquella que mantiene unas buenas relaciones familiares? El Cociente Intelectual, ¿determina nuestro destino? El concepto de inteligencia humana debe ir más allá y englobar un amplio abanico de capacidades esenciales para la vida, que Daniel Goleman llama Inteligencia Emocional .

Este nuevo concepto de inteligencia explicaría el porqué personas con un alto cociente intelectual fracasan en aspectos básicos de sus vidas y en cambio personas con un CI no tan alto triunfan en nuestra sociedad. La inteligencia emocional tendrá en cuenta tanto los sentimientos como

las habilidades para el control de los impulsos, la autoconciencia, la motivación, el entusiasmo, la perseverancia, la empatía, la agilidad mental, etc.

En palabras de Daniel Goleman, "toda persona posee inteligencia cognitiva e inteligencia emocional, aunque lo cierto es que la inteligencia emocional aporta, con mucha diferencia, la clase de cualidades que más nos ayudan a convertirnos en auténticos seres humanos" ([**Goleman, 1998**]).

A.2.4 ¿Heredada o Aprendida?

Una pregunta que nos planteamos al hablar de inteligencia es si la persona inteligente "nace o se hace". ¿La inteligencia la heredamos de nuestros padres al igual que muchas características del aspecto físico? o ¿la inteligencia la desarrollamos en los primeros años dependiendo del ambiente en que nos criemos?

Durante mucho tiempo prevaleció la idea de que la inteligencia era hereditaria, y así se hablaba de familias de genios (la dinastía de músicos Bach, los matemáticos Bernoulli, ...): "de padres listos nacen hijos listos". Estudios efectuados con niños adoptados han demostrado que el cociente intelectual de niños de padres con bajo C.I., situados en medios de adopción favorables socioculturalmente, eran más parecidos al de los padres adoptivos que los biológicos. También se han realizado estudios con gemelos univitelinos (idéntica carga genética), de forma que los gemelos criados juntos presentan una elevadísima correlación en los resultados de los tests de inteligencia, y en los pocos casos en que se han educado por separado, aunque existe una alta similitud, era algo inferior.

Pero el desarrollo intelectual depende de numerosos factores y actualmente nadie pone en duda que la inteligencia humana es el resultado de la herencia genética y del ambiente en que se desarrollan los primeros años de la vida. Hay gran cantidad de investigaciones en psicología que corroboran esta idea. Tanto los genes como el ambiente configuran la inteligencia.

Parece evidente que el cerebro necesita estimularse y ejercitarse para madurar. Estudios realizados demuestran que hay edades para aprender cada una de las habilidades (*periodo sensible*). Más allá de estos periodos ni los estímulos ni el aprendizaje serán suficientes para estimular una función del sistema nervioso.

Pero nos podemos preguntar ¿hasta que punto debemos hiperestimular y manipular el ambiente para incrementar su influencia sobre la inteligencia? Según algunos psicólogos la sobreestimulación es buena para

compensar los efectos de un ambiente empobrecido, pero no garantizan mejoras en un ambiente bueno. En palabras de la psicóloga Pilar Varela "una formación demasiado intensa y extraordinaria puede hacer a una persona más culta, pero no es seguro que la haga más inteligente y tal vez la haga más desgraciada".

A.2.5 Aspectos de la Inteligencia

Muchos son los aspectos que intervienen en lo que venimos llamando inteligencia. Todos son importantes como procesos del pensamiento. Aquí vamos a enumerar brevemente algunos de ellos:

- La memoria
- El pensamiento abstracto y el razonamiento
- El lenguaje y la comunicación
- El aprendizaje
- La resolución de problemas
- La creatividad

En este documento se han tratado algunos de ellos. Otros vamos a comentarlos brevemente aquí.

La memoria

Podemos aprender infinidad de cosas, pero ¿de qué nos servirían si no las recordásemos? El cerebro humano puede almacenar billones de datos. Sin la memoria no conoceríamos nuestra propia historia ni podríamos retener lo aprendido. La memoria es una doble habilidad que nos permite guardar nuevos recuerdos así como recuperar otros almacenados con anterioridad. La memoria es un sistema con tres componentes que funcionan de manera integral:

- la *memoria sensorial* (MS) que retiene durante unos instantes (menos de 2 segundos) la información exterior captada por los sentidos, generalmente de forma inconsciente
- la *memoria a corto plazo* (MCP) que retiene la información que nos interesa durante un periodo más largo (unos 20 segundos) para pensar conscientemente y valorar su interés,

- y la *memoria a largo plazo* (MLP) donde permanece (minutos, días o toda la vida) esa información que conscientemente hemos transferido a este sistema.

El lenguaje y la comunicación

En este momento la inteligencia es estructuralmente lingüística, ya que gracias al lenguaje no sólo aprovechamos los conocimientos de las generaciones anteriores, sino también sus habilidades y destrezas psicológicas. Sabemos que lenguaje e inteligencia se refuerzan. Lo que no se sabe aún es si los humanos evolucionaron y lograron utilizar tan bien el lenguaje debido a que son inteligentes, o que son inteligentes debido a que saben utilizar el lenguaje bien.

El aprendizaje

El ser humano llega a este mundo totalmente ignorante. Pero desde el instante en que empieza a recibir sensaciones su actividad neuronal empieza a establecer patrones. El aprendizaje no sólo nos permite adquirir habilidades y conocimientos; también permite modificar nuestra conducta para adaptarnos continuamente al medio en que vivimos.

A.3 Medidas de la Inteligencia

Hemos asumido que las personas somos inteligentes en distinto grado y por ello podremos querer medir esa inteligencia. Respecto a las mediciones de la inteligencia tampoco hay acuerdo. Los primeros intentos de medir aspectos psicológicos para clasificar a las personas datan de finales del siglo pasado y se deben a Francis Galton. Ideó un sistema de medición mental, aunque rápidamente quedo desfasado. Pero se le considera el precursor de la idea de que la capacidad mental se puede medir. Los siguientes paso en la investigación de la medida de la inteligencia fueron dados por el psicólogo Alfred Binet en el ámbito escolar, que consideraba que un niño inteligente debería rendir como otro de más edad, y un niño con poca inteligencia como otro de menor edad. Así, surge el concepto de *edad mental*, base del cociente intelectual que trataremos más adelante.

A.3.1 Los Tests de Inteligencia

Aunque no hayamos definido claramente qué es inteligencia, sí que sabemos que existen unas personas más inteligentes que otras, por lo que

querremos medirla de alguna manera. La inteligencia se puede manifestar de muchas maneras: obteniendo buenas calificaciones en los estudios, ganando al participar en un juego de estrategia, ascendiendo en el trabajo, ... Pero una forma de evaluarla de manera científica es a través de los *tests de inteligencia.* Aunque no son infalibles, si se hacen bien las cosas, los tests de inteligencia constituyen un instrumento eficaz para establecer objetivamente la capacidad mental (y evitar establecer diferencias de otro tipo, como el sexo, procedencia social, aspecto físico, ...).

El concepto *medir* significa comparar y por ello para medir la inteligencia de una persona debemos compararla con la inteligencia de las otras personas. Así, si decimos que nuestra inteligencia es "normal" estamos diciendo que es similar a la mayoría de la gente. Tanto los superdotados como las personas con un nivel intelectual muy bajo son una minoría. Esto tiene una representación matemática en la llamada *distribución normal* o *campana de Gauss* (figura A.1).

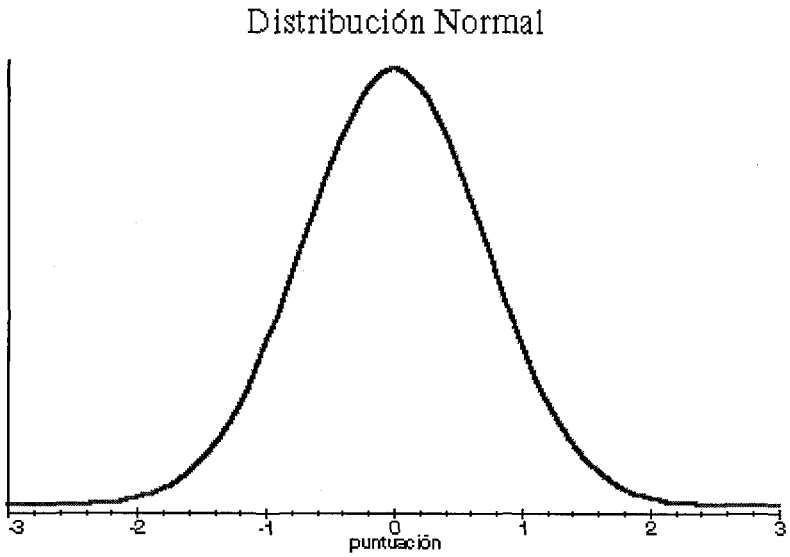


Figura A.1: Representación Gráfica de una Distribución Normal

Esta gráfica la podemos interpretar sabiendo que en el eje x (línea horizontal) colocamos la puntuación en el test de inteligencia y en el eje y (línea vertical) el número de personas que obtienen esa puntuación. Vemos claramente que hay muy pocas personas superinteligentes, así como también hay muy pocas personas con baja puntuación. La gran mayoría (el 68%) se encuentran en niveles muy cercanos (entre -1 y 1) y el 95% se encuentran entre -2 y 2.

Los tests están sometidos a una rigurosa investigación y control estadístico. Deben ser:

válido : mide lo que se pretende medir

fiable : siempre mide lo mismo

Otro aspecto es que es difícil elaborar tests que no contengan sesgo cultural. Además, los tests deben complementarse con otras pruebas de valoración de la inteligencia. Los psicólogos deben efectuar varios tests y otras valoraciones que refrenden el conjunto de resultados obtenidos. Un requisito fundamental para la aplicación de un test es su riguroso control por un profesional, que selecciona las pruebas adecuadas a la finalidad pretendida, da una explicación clara de las instrucciones y del tiempo disponible y garantiza unas condiciones correctas de ejecución de la prueba y de evaluación de los resultados.

A.3.2 El Cociente Intelectual

Los resultados de los tests de inteligencia vienen expresados en términos del *cociente intelectual* (C.I.). Según Binet la medida de la inteligencia se puede obtener dividiendo la edad mental (E.M.), obtenida por las pruebas de inteligencia, entre la edad cronológica de la persona (E.C.), y multiplicando por 100:

$$C.I. = \frac{E.M.}{E.C.} \cdot 100$$

Podemos ver fácilmente que la persona que posee un cociente intelectual de 100 posee una inteligencia normal (cociente intelectual medio), es decir, tiene la misma edad mental que cronológica. Esta interpretación del cociente intelectual, que funciona con los niños, no funciona con los adultos. Desde final de la adolescencia, la inteligencia no aumenta (se "para") a diferencia de la edad cronológica. Es más, un adulto de 50 años que rinde como un joven de 20 años, ¿qué cociente intelectual tiene? Vemos por tanto que debemos dar otra significación a los resultados de los tests de inteligencia. Así, el cociente intelectual (C.I.) lo podemos interpretar como:

- para niños, la relación de la edad mental con la real.
- para adultos, el índice de la superioridad (o inferioridad) de su capacidad intelectual con relación a individuos de su mismo grupo de edad.

Otro de los problemas planteados por el C.I. es su estabilidad, ya que representa una velocidad de desarrollo, que se supone constante, cuando la observación nos dice que no es así.

A.3.3 El Test de Turing

Alan Turing declaró que "puede considerarse que una máquina es inteligente si puede pasar por un ser humano en una prueba ciega" y así, en su histórico articulo de 1950 propuso el *juego de imitación* (más conocido como el *Test de Turing* [1]) para determinar razonablemente si una máquina piensa:

Un ordenador y un humano se ocultan a la vista del interrogador; éste debe de tratar de averiguar cuál es el ordenador y cuál es el ser humano mediante el planteamiento de preguntas a cada uno de ellos; el humano responderá a las preguntas sinceramente tratando de persuadirle de que él es realmente el ser humano y el otro es el ordenador, y la computadora está programada para mentir intentando convencer al interrogador de que es el ser humano y el otro la máquina. Si el interrogador es incapaz de identificar de una forma definitiva al ser humano real, el ordenador ha superado la prueba.

Varias han sido las objeciones que se le han puesto al test de Turing. Entre ellas destacar la del filósofo John Searle conocido como la *Sala China* [2]:

Supongamos que tenemos un sistema que pasa la prueba de Turing. Dicho sistema podría semejarse a un sistema formado por un humano que únicamente entiende el castellano encerrado en una habitación con un libro de reglas. Mediante una apertura recibe del exterior unas papeletas con símbolos indescifrables, pero que localizados en el libro de reglas le permiten ejecutar una serie de instrucciones. La salida final es devuelta al exterior. Visto desde el exterior este sistema trabaja con oraciones escritas en chino como entrada y produce una respuesta también en chino. Pero, si el humano no entiende el chino, el libro de reglas no entiende chino ¿aquí nadie comprende el chino?

---

[1] Descrito por primera vez en el artículo "Computing Machinery and Intelligence" de Alan M. Turing publicado en 1950 en la revista "Mind"

[2] Descrito por primera vez en el artículo "Minds, Brains, and Programs" de John R. Searle publicado en 1980 en la revista "The Behaviorial and Brain Sciences" 3, pág. 417-424.

Más información, así como una reimpresión de estos artículos se puede encontrar en [**Luger, 1995**].

A.4 Inteligencia Artificial

Está de moda hablar de inteligencia. Aunque para muchos la inteligencia sería la característica que distingue al hombre de las demás especies, podemos decir que la inteligencia ya no es sólo una cualidad humana, o de forma más general de los seres vivos (se puede considerar que los animales son inteligentes). Las nuevas máquinas poseen (o se dice que poseen) esta cualidad, y por ello se habla de lavadoras inteligentes, edificios inteligentes, etc. Pero ¿son realmente inteligentes? ¿pueden pensar las máquinas? Se nos venden muchos productos como inteligencia artificial, pero ¿qué es realmente la inteligencia artificial?

A los ordenadores en un principio se les llamaba cerebros electrónicos

La Inteligencia Artificial (IA), definida usualmente como "el estudio de cómo programar computadoras que posean la facultad de hacer aquello que la mente humana puede realizar" (Minsky), toma un sentido científico viable, como disciplina Informática moderna, durante la segunda mitad de este siglo, y es el resultado directo de la confluencia de diversas corrientes intelectuales (Teoría de la Computación, Cibernética, Teoría de la Información, Procesamiento Simbólico) desarrolladas sobre los cimientos formales de la Lógica y la Matemática Discreta, e impulsadas por el desarrollo de los ordenadores digitales.

A.4.1 Introducción

En los años cincuenta y sesenta los avances tecnológicos de los ordenadores impulsaron un resurgimiento de las investigaciones psicológicas de la mente humana. El autómata de estados finitos de Alan Turing, la teoría del pensamiento en términos mecanicistas de Craik ([**Craik, 1943**]), el desarrollo de la máquina de John von Neumann, la teoría de la información de Shannon ([**Shannon, 1948**]), el desarrollo de un sistema cibernético por el matemático Norbert Wiener ([**Wiener, 1998**]) y la visión de las neuronas como unidades lógicas portadoras de información de McCulloch y Pitts ([**McCulloch and Pitts, 1948**]), entre otros estudios, sentaron los cimientos de la psicología cognitiva ([**Wilson and Keil, 1999**]). Así se empieza a pensar en la mente como un sistema activo que procesa información. Esto abrió las puertas para que se investigase sobre la posibilidad de que el pensamiento pudiera automatizarse. Para ello necesitamos modelos computacionales de como trabaja el pensamiento ([**Hofstadter, 1995**]).

En 1955 McCarthy acuñó el termino *Inteligencia Artificial* para englobar todas las actividades encaminadas a la construcción de sistemas inteligentes, así la IA estudia como lograr que las máquinas realicen tareas que, por el momento, son realizadas mejor por los seres humanos: la comprensión del lenguaje, el aprendizaje, el razonamiento, la resolución de problemas, etc. Se observó que los expertos humanos tienen un dominio de experiencia muy restringido y se comenzó el desarrollo de sistemas que presentasen comportamientos inteligentes en dominios muy limitados; así se desarrollaron distintos sistemas expertos en diversos campos del saber: química, medicina y geología, entre otros. Podemos considerar a la Inteligencia Artificial aplicada como un intento de modelar el conocimiento humano, de forma más o menos similar a como otras ciencias (como la matemática, la física o la química) han modelo al mundo real.

A lo largo de la historia se han adoptado varios enfoques para definir la IA. Podríamos decir que las diversas definiciones se agrupan en cuatro categorías:

- sistemas que piensan como humanos
- sistemas que actúan como humanos
- sistemas que piensan racionalmente
- sistemas que actúan racionalmente

La inteligencia artificial no puede competir con la estupidez natural

Seguidamente, presentamos algunas definiciones de lo que es la IA de acuerdo con algunos autores destacados:

El estudio de las facultades mentales mediante el uso de modelos computacionales (Charniak y McDermott).

La inteligencia artificial es el estudio de los cálculos que permiten percibir, razonar y actuar (Winston).

La Inteligencia Artificial (IA) estudia cómo lograr que las máquinas realicen tareas que, por el momento, son realizadas mejor por seres humanos (Rich y Knigth).

La IA es la ciencia de construir máquinas para que hagan cosas que, si las hicieran los humanos, requerirían inteligencia (Minsky).

La IA se presenta en diversas áreas en mayor o menor medida. A continuación comentamos brevemente algunas de estos campos que conformarían la Inteligencia Artificial:

- *Tratamiento de lenguajes naturales*: En este campo se pueden englobar aplicaciones que realicen traducciones entre idiomas, interfaces hombre-máquina que permitan interrogar una base de datos o dar órdenes a un sistema operativo, etc, de manera que la comunicación sea más amigable al usuario.
- *Sistemas Expertos*: Aquí se incluyen aquellos sistemas donde la experiencia de personal cualificado se incorpora a dichos sistemas para conseguir deducciones más cercanas a la realidad.
- *Robótica*: Navegación de robots móviles, control de brazos de robots, ensamblaje de piezas, etc.
- *Problemas de percepción: visión y habla.* Reconocimiento de objetos y del habla, detección de defectos en piezas por medio de visión, apoyo en diagnósticos médicos, etc.
- *Aprendizaje*: Modelización de conductas para su posterior implantación en computadoras.

Vamos a aclarar algunos conceptos que nos permitirán seguir la terminología utilizada, tendríamos:

- *IA fuerte*: tener una mente consiste en tener un programa
- *IA débil*: los procesos cerebrales (y los procesos mentales) se pueden simular computacionalmente
- *Cognitivismo*: el cerebro es un ordenador digital
- *Funcionalismo*: los estados mentales en general no son otra cosa que estados computacionales matemáticamente definidos
- *Dualismo*: el universo contiene dos tipos de elementos muy diferentes, el elemento mental y el elemento físico, y ambos existen en forma semiindependientes el uno del otro
- *Monismo*: afirma que en realidad existe sólo un tipo de elemento, del cual están hechos, en última instancia, tanto las mentes como los cerebros
- *Fisicismo*: las sensaciones subjetivas particulares son en realidad idénticas a los procesos físicos cerebrales

La **Inteligencia Artificial fuerte** ("strong AI") aboga porque una máquina programada adecuadamente puede tener procesos mentales. La **Inteligencia Artificial débil** ("weak AI") dice que las máquinas pueden simular los procesos mentales. Aquí no entraremos en esa polémica, ni si los ordenadores tienen conciencia. Tras años de trabajos en Inteligencia Artificial (IA) y expectativas, más o menos cumplidas o fallidas, lo que parece evidente es que los sistemas complejos necesitan cada vez más de software inteligente. Por tanto en este trabajo no se ha pretendido construir máquinas con habilidades humanas, sino el desarrollo de técnicas de IA que hagan el software más intuitivo y fácil de usar, al mismo tiempo que más productivo.

Los cerebros son computadores biológicos

Lo que es evidente es que los programas y técnicas informáticas de la Inteligencia Artificial apoyan a los científicos en las investigaciones de los procesos mentales. Y así, el avance y desarrollo de los ordenadores ha influido significativamente de dos maneras, distintas y al mismo tiempo relacionadas:

- los ordenadores proporcionan una herramienta para probar nuevas teorías
- y al mismo tiempo han motivado la aparición de la analogía entre cerebros y computadores, de forma que se propone que los procesos mentales son a los cerebros lo que los programas son a los ordenadores sobre los que se ejecutan

Una buena panorámica de la evolución histórica de la inteligencia artificial la encontraremos en [**Crevier, 1996**]. Un ataque a la IA fuerte aparece en [**Penrose, 1991**]. Libros sobre Inteligencia Artificial, entre otros, son [**Charniak and McDermott, 1985**], [**Rich and Knight, 1994**], [**Winston, 1994**] y [**Luger and Stubblefield, 1999**].

A.4.2 Ingeniería del Conocimiento

A la parte de la Inteligencia Artificial dedicada al estudio de los principios, métodos y herramientas aplicables a la utilización del conocimiento humano en materias concretas y de sus fuentes para construir sistemas informáticos que reflejen ese conocimiento se le denomina *Ingeniería del Conocimiento* ([**Palma *et al.*, 2000**]).

Durante los años 70 se produjeron una serie de desarrollos por parte de los investigadores de Inteligencia Artificial que dieron lugar al concepto de *Sistemas Basados en Conocimiento* (capítulo 3), que no sólo deben facilitar los hechos que han ocurrido previamente, sino que deben ser

capaces de extraer conclusiones e incorporar nuevos hechos para poder ser utilizados en posteriores razonamientos. Estos sistemas proponían, para resolver los problemas de un determinado tipo, la organización de la aplicación en 2 componentes :

Base de Conocimiento : componente declarativa en la que aparece de forma explícita lo que debe conocer un agente inteligente especializado en ese tipo de problemas.

Proceso de Razonamiento : procedimiento diseñado para que partiendo de la pregunta, considerada como premisa, llegar a las posibles respuestas que se derivan de la base de conocimiento.

Estas dos componentes determinan los dos primeros capítulos, ya que necesitamos por una lado un modelo (o lenguaje) de representación del conocimiento (capítulo 1) que tenemos en un dominio y por otro lado, trabajaremos con un sistema de razonamiento (capítulo 2) que nos permitirá extraer información nueva a partir de la que ya tenemos. Es necesario contar con la capacidad para representar el conocimiento y razonar en base a él, pues de esta manera se podrán tomar decisiones correctas en una amplia gama de situaciones.

Esta organización de la información es más comprensible por el usuario (no programador) que los pasos de un proceso algorítmico formulado en algún lenguaje de programación convencional. Además, este tipo de arquitecturas proporciona la posibilidad de incluir en la base de conocimiento tanto elementos de teoría del dominio de tipo de problemas estudiado como criterios heurísticos derivados de la experiencia de las personas. De ahí que en una primera época se denominaran *Sistemas Expertos*. Los sistemas expertos son sistemas basados en el conocimiento dedicados a tareas específicas que requieren una gran cantidad de conocimiento de un dominio de experiencia especializado. Proporcionan experiencia en forma de diagnósticos, instrucciones, predicciones o consejos ante situaciones reales que se le planteen. Se puede pensar en ellos, más que como sustitutos del experto humano, como un consultor que puede proporcionar una ayuda al experto humano en la toma de decisiones.

Apéndice B

SINTAXIS

> La elección de una notación constituye una etapa importante en la solución de un problema. Debe elegirse con cuidado. /.../ Una notación apropiada podrá contribuir de modo primordial a la comprensión del problema.
>
> *How to solve it.* Polya

En este anexo describiremos brevemente la sintaxis de algunos lenguajes utilizados para el desarrollo del proyecto, como complemento y ayuda del texto. Únicamente tratamos los lenguajes específicos del ámbito del trabajo: CLIPS/JESS para la escritura de las reglas (nuestros ficheros de conocimiento) y KQML para la implementación del protocolo de comunicación entre los agentes para el intercambio de dicho conocimiento.

B.1 CLIPS y JESS

JESS ("Java Expert System Shell") es una herramienta implementada en Java que va a permitir introducir razonamiento basado en reglas en cualquier programa desarrollado a partir del lenguaje Java. Básicamente, Jess es un intérprete de un lenguaje de reglas similar al CLIPS ("C Language Integrated Production System"). A continuación, se describen pequeños rasgos del lenguaje. Para una definición más exhaustiva, es aconsejable consultar el manual de Jess y/o de Clips.

Átomos

Un átomo o símbolo es el concepto básico del lenguaje Jess. Un átomo puede estar formado por letras, números y algunos caracteres especiales ($*=+/<>_?#), pero no puede empezar por un número.

Cadenas

Las cadenas se delimitan con comillas dobles (""). Para usar caracteres de escape, se utiliza la barra (\).

Listas

La lista es la unidad fundamental en Jess. Una lista se compone de un conjunto de átomos, números, cadenas, otras listas, encerrados entre paréntesis.

Comentarios

Todo lo que se escriba detrás de un punto y coma (;) hasta el final de la línea será ignorado por el intérprete.

Funciones

En Jess, las funciones siguen la notación prefija. Por tanto, una lista cuyo primer elemento sea un nombre de una función, se evaluará como expresión. En el manual de Jess se encuentran todas las funciones enumeradas alfabéticamente.

Variables

Las variables en Jess son átomos cuyo nombre empieza por el carácter de interrogación (?). Si, además, el nombre comienza por el carácter de dólar ($), la variable es una *multivariable*, que indica que se trata de una lista que puede contener varios valores. En este caso, a la lista se le llama *multicampo*. Para asignar un valor a una variable se utiliza la función `bind`, y en el caso de multicampos, se deben crear con la función `create$`. Si se quiere utilizar una variable global se debe utilizar el constructor `defglobal` para definir e inicializar la variable.

Funciones definidas por el usuario

Se pueden definir funciones mediante el constructor `deffunctions`, al cual se le indica el nombre de la función, un comentario explicativo, los parámetros y el cuerpo de la función, pudiendo devolver algún valor.

Hechos

Los hechos definen el estado actual del sistema. Pueden definirse hechos con el nombre de un átomo, o con el nombre de una plantilla (`deftemplate`) que contiene distintos parámetros (similar a las estructuras de otros lenguajes). Para colocar un hecho en la lista de hechos, se utiliza la función `assert`; para eliminarla, se utiliza `retract`. Para ver la lista actual de hechos, hay que llamar a la función `facts`. Mediante el atributo `salience` de los hechos, éstos pueden tener asociada una prioridad (un número entero). Cuanto menor sea el entero, menos prioridad

tiene el hecho. Si se quiere almacenar hechos en la lista antes de que el programa Jess se ejecute, es necesario utilizar el constructor `deffacts`.

Plantillas

Se pueden especificar plantillas ("templates") que almacenen más de un parámetro. Una plantilla se divide en "slots" (o "multislots", si los parámetros son multicampo). A cada slot se le puede asignar un valor por defecto, bien estático, bien como resultado de aplicar una función.

Clases

Jess permite convertir un Java Bean en una plantilla a través del constructor `defclass`. Asimismo, los `deffacts` son a `deftemplates` como `definstance` son a `defclass`.

Reglas

Es la parte importante de Jess, la finalidad última de la ejecución del programa. La reglas en Jess son similares a construcciones del tipo *if...then* en otros lenguajes. Internamente, Jess comprueba si alguna parte de los *if* se verifica, y ejecuta en consecuencia los respectivos *then*. Si una parte *if* (parte izquierda de la regla, LHS) se evalúa a cierto, la regla se activará, pero puede que no se dispare (esto es, no se ejecute la parte *then* o parte derecha de la regla, RHS). Es importante diferenciar cuándo una regla se *activa* (la regla se evalúa a cierto, pero la parte derecha aún no se ha ejecutado) y cuándo se *dispara* (la regla se ejecuta, con lo que no va a activarse otra vez). El caso más típico es introducir mediante `assert` hechos que activen las distintas reglas, pero que no se dispararán mientras no se llame a la función `run`.

B.2 KQML

KQML ("Knowledge Query and Manipulation Language") es un lenguaje y protocolo para el intercambio de información y conocimiento, surgido de los trabajos para el desarrollo de técnicas y metodologías para la construcción de bases de conocimiento a gran escala que sean compartidas y reusables. La sintaxis de KQML es similar a la de LISP (notación en forma de listas de paréntesis balanceados).

Todo mensaje en KQML se inicia con una performativa que indica el tipo de comunicación y un conjunto de argumentos opcionales, llamados *parámetros*. Los nombres de los parámetros empiezan con ":" y deben estar seguidos por el correspondiente *valor del parámetro*. Los posibles parámetros con su correspondiente descripción se pueden ver en la tabla

B.1. El conjunto de performativas estándar con su significado aparece en las tablas B.2 y B.3. Representamos con S el `:sender`, con R el `:receiver` y con VKB ("Virtual Knowledge Base") la base de conocimiento virtual.

Palabra clave	Descripción
`:sender`	emisor (actual) de la performativa
`:receiver`	receptor (actual) de la performativa
`:from`	el origen (emisor virtual) de la performativa de `:content` cuando se hace un `forward`
`:to`	el destinatario final (receptor virtual) de la performativa de `:content` cuando se hace un `forward`
`:in-reply-to`	etiqueta esperada en la respuesta al mensaje previo (la misma de `:reply-with` del mensaje previo)
`:reply-with`	etiqueta esperada en la respuesta al actual mensaje
`:language`	lenguaje de representación en el que está escrito `:content`
`:ontology`	nombre de la ontología asumida en `:content`
`:content`	information con la que la performativa expresa una actitud

Tabla B.1: *Parámetros* Reservados para las *Performativas* KQML

Performativa	Descripción
ask-if	S quiere saber si :content está en la VKB de R
ask-all	S quiere todas las instanciaciones de R en los que :content se cumple
ask-one	S quiere una instanciación de R en la que :content se cumple
stream-all	version de respuesta multiple de ask-all
eos	"end-of-stream" para una respuesta multiple (stream-all)
tell	la sentencia está en la VKB de S
untell	la sentencia no está en la VKB de S
deny	la negación de la sentencia está en la VKB de S
insert	S pide a R que añada :content en su VKB
uninsert	S pide a R que deshaga un insert previo
delete-one	S pide a R que quite una sentencia que empareje de su VKB
delete-all	S pide a R que quite todas las sentence que emparejen de su VKB
undelete	S pide a R que deshaga un delete previo
achieve	S pide a R que haga se cumpla algo en su entorno físico
unachieve	S pide a R que deshaga un achieve previo
advertise	S pide a R saber si S puede y procesará un mensaje como :content
unadvertise	S pide a R saber si S cancelará un advertise previo y no procesará ningún mensaje como :content
subscribe	S quiera actualizar la respuesta de R como una performativa
error	S considera que el anterior mensaje de R estaba mal formado
sorry	S entiende el mensaje de R pero no puede proporcionarle información
standby	S pide a R que anuncie su disposición a proporcionar respuesta al mensaje en :content
ready	S está preparado para responder el mensaje previo recibido de R
next	S pide la siguiente respuesta de R al mensaje previo mandado por S
rest	S pide las respuestas de R que queden al mensaje previo mandado por S
discard	S no quiere las respuestas de R que queden al mensaje previo.

Tabla B.2: *Performativas* KQML Reservadas (1)

Volver al índice/Tornar a l'índex

Performativa	Descripción
`register`	S notifica a R su presencia y su nombre
`unregister`	S pide a R deshacer el `register` previo
`forward`	S pide a R que redirija el mensaje al `:to` agente
`broadcast`	S pide a R que mande un mensaje a todos los agentes que R conozca
`transport-address`	S asocia su nombre con una nueva dirección de transporte
`broker-one`	S pide a R que encuentre una respuesta a <performative> (otro agente distinto de R puede proporcionar esa respuesta)
`broker-all`	S pide a R que encuentre todas las respuestas a <performative> (otro agente distinto de R puede proporcionar esa respuesta)
`recommend-one`	S quiere averiguar un agente que pueda responderle a <performative>
`recommend-all`	S quiere averiguar todos los agentes que puedan responderle a <performative>
`recruit-one`	S pide a R obtenga un agente apropiado que responda a <performative>
`recruit-all`	S pide a R obtenga todos los agentes apropiados que respondan a <performative>

Tabla B.3: *Performativas* KQML Reservadas (2)

ÍNDICE DE MATERIAS

Volver al índice/Tornar a l'índex