

T24624

I

UNIVERSIDAD COMPLUTENSE  
DE  
MADRID



FACULTAD DE BIOLOGIA

DEPARTAMENTO DE MATEMATICA APLICADA (BIOMATEMATICA)



UNIVERSIDAD COMPLUTENSE



5314061528

NUEVAS TECNICAS DE MODELADO  
ORIENTADO A OBJETOS  
E IMPLEMENTACION DE UN  
GENERADOR DE SISTEMAS BASADOS  
EN CONOCIMIENTO

TESIS DOCTORAL

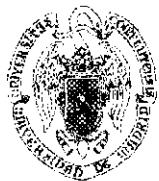
M<sup>a</sup> de la Almudena Bailador Ferreras

Madrid, 2000



2000-07-06

**UNIVERSIDAD COMPLUTENSE  
DE  
MADRID**



**FACULTAD DE BIOLOGIA**

**DEPARTAMENTO DE MATEMÁTICA APLICADA (BIOMATEMÁTICA)**

**NUEVAS TECNICAS DE MODELADO  
ORIENTADO A OBJETOS  
E IMPLEMENTACION DE UN GENERADOR DE SISTEMAS  
BASADOS EN CONOCIMIENTO**

Memoria presentada  
para optar al grado de Doctor

A handwritten signature in black ink.

Fdo. M<sup>a</sup> de la Almudena Bailador Ferreras

Vº Bº Director de la tesis

A handwritten signature in black ink.

Fdo. Julio Alonso Fernández

Madrid, 2000

*A mis padres*

*A Emilio*

# ÍNDICE

---

## TOMO I

Agradecimientos .....	viii
Lista de figuras .....	xi
Lista de tablas .....	xii

### I INTRODUCCIÓN

<b>1. Aspectos generales</b>	
1.1. Motivación y objetivos .....	2
1.2. Presentación .....	4
1.3. Organización .....	7
<b>2. Sistemas Basados en Conocimiento</b>	
2.1. Fundamentos generales .....	10
2.2. Adquisición del conocimiento .....	17
2.3. Representación del conocimiento y razonamiento.....	18
2.4. Control de la coherencia.....	20
2.5. Propagación de la incertidumbre.....	22
2.5.1. Probabilidad subjetiva .....	23
2.5.2. Factores de certeza .....	26
2.5.3. Redes bayesianas.....	30
<b>3. Tecnología de Objetos</b>	
3.1. Orígenes de la Orientación a Objetos .....	34
3.2. Evolución de lenguajes .....	36
3.3. Evolución de metodologías .....	38
3.4. Metodologías orientadas a objetos.....	41
3.4.1. Metodología de Coad-Yourdon.....	42
3.4.2. Metodología de Booch.....	43
3.4.3. Metodología de Wirfs-Brock .....	43
3.4.4. Metodología de Jacobson.....	44
3.4.5. Metodología de Rumbaugh.....	45
3.4.6. Lenguaje unificado de modelado .....	50

3.5. Patrones orientados a objetos .....	53
3.5.1. Origen y fundamentos generales .....	53
3.5.2. Clasificación de patrones .....	55
3.5.3. Lenguajes de patrones .....	56
3.5.4. Patrones de diseño.....	57

## **II ELABORACIÓN DE UN LENGUAJE DE PATRONES**

### **4. Fundamentos generales**

4.1. Patrón evolutivo de la naturaleza.....	61
4.2. Presentación del lenguaje de patrones .....	67
4.2.1. Nombre .....	68
4.2.2. Objetivo .....	68
4.2.3. Motivación.....	69
4.2.4. Aplicación.....	70
4.2.5. Estructura.....	70
4.2.6. Participantes y colaboraciones .....	75

### **5. Lenguaje de patrones para la Síntesis y Regulación de Objetos Visuales**

5.1. Modelo de Síntesis de Objetos Visuales.....	77
5.1.1. Nombre y Objetivo.....	77
5.1.2. Estructura.....	79
5.1.3. Participantes y colaboraciones .....	95
5.1.4. Aplicación.....	95
5.2. Modelo de Regulación de Objetos Visuales.....	97
5.2.1. Nombre y Objetivo.....	97
5.2.2. Estructura.....	98
5.2.3. Participantes y colaboraciones .....	98
5.2.4. Aplicación.....	99
5.3. Refinamiento del Modelo de Síntesis de Objetos Visuales .....	101
5.3.1. Nombre y Objetivo.....	101
5.3.2. Estructura.....	102
5.3.3. Participantes y colaboraciones .....	105
5.3.4. Aplicación.....	105
5.4. Refinamiento del Modelo de Regulación de Objetos Visuales.....	107
5.4.1. Nombre y Objetivo.....	107
5.4.2. Estructura.....	107
5.4.3. Participantes y colaboraciones .....	109
5.4.4. Aplicación.....	109

## **III ELABORACIÓN DE UN PATRÓN DE ARQUITECTURA**

### **6. Presentación del patrón de arquitectura**

6.1. Planteamiento del problema.....	112
6.2. Requisitos .....	113
6.3. Solución.....	113

<b>7. Subsistema de desarrollo</b>	
7.1. Representación del conocimiento .....	116
7.1.1. Representación con reglas de producción.....	117
7.1.2. Representación con marcos de clasificación.....	119
7.1.3. Representación con marcos causales.....	120
7.2. Validación y verificación de la base de conocimiento .....	121
7.2.1. Aspectos generales .....	121
7.2.2. Estructura interna .....	121
7.2.3. Funcionamiento en reglas de producción. ....	124
7.2.4. Funcionamiento en marcos de clasificación. ....	126
7.2.5. Funcionamiento en marcos causales. ....	128
7.3. Representación de la incertidumbre .....	129
7.3.1. Aspectos generales .....	129
7.3.2. Funcionamiento con probabilidad subjetiva .....	129
7.3.3. Funcionamiento con factores de certeza.....	134
7.3.4. Funcionamiento con redes bayesianas.....	134
<b>8. Subsistema de ejecución</b>	
8.1. Presentación del conocimiento.....	137
8.1.1. Arboles de toma de decisión.....	137
8.1.2. Redes de conceptos .....	139
8.2. Propagación del conocimiento.....	142
8.2.1. Funcionamiento general .....	142
8.2.2. Arboles de toma de decisión.....	145
8.2.3. Redes de conceptos .....	147
8.3. Propagación de la incertidumbre.....	149
8.3.1. Probabilidad subjetiva .....	149
8.3.2. Factores de certeza .....	150
8.3.3. Redes bayesianas.....	154
<b>IV DESARROLLO INFORMÁTICO</b>	
<b>9. Integración del lenguaje de patrones con el patrón de arquitectura</b>	
9.1. Captura de requisitos.....	158
9.1.1. Fichas descriptivas.....	159
9.1.2. Fichas Conceptuales .....	160
9.1.3. Fichas Funcionales.....	161
9.1.4. Fichas Visuales.....	162
9.1.5. Extracción de elementos .....	163
9.2. Subsistema de desarrollo .....	164
9.2.1. Representación del conocimiento.....	164
9.2.2. Validación y verificación de la base de conocimiento.....	165
9.2.3. Representación de la incertidumbre.....	167
9.3. Subsistema de ejecución .....	168
9.3.1. Presentación del conocimiento .....	168
9.3.2. Propagación del conocimiento.....	169
9.3.3. Propagación de la incertidumbre .....	169

**10. Implementación**

10.1. Descripción informática .....	173
10.2. Desarrollo informático .....	175
10.2.1. Subsistema de desarrollo.....	175
10.2.2. Subsistema de ejecución.....	183
10.2.3. Pseudocódigos.....	185

**V CONCLUSIONES****11. Resultados obtenidos**

11.1. Aportaciones de la tesis.....	189
11.2. Líneas futuras de desarrollo.....	192

**BIBLIOGRAFÍA****ANEXO (TOMO II)**

## **AGRADECIMIENTOS**

---

La elaboración de una tesis no es fácil, pero si, además, es necesario simultanearla con el trabajo, conlleva un esfuerzo sometido a frecuentes desánimos ya que el tiempo a invertir forma parte de los momentos que uno, en condiciones normales, considera de ocio, arrastrando consigo no sólo el descanso propio sino el de la familia, los amigos y más tarde el de la pareja, como ocurre en mi caso. Esta tesis es el fruto de una serie de años de esfuerzo, de privaciones y en muchos casos de sacrificios, que he podido superar porque en todo momento me he sentido acompañada y apoyada por diferentes personas, tanto en el ámbito profesional como en el personal, fundamentalmente.

Mi primer agradecimiento debe ir destinado al director de esta tesis. Julio Alonso, a quien debo el inicio en la investigación y ser sobre todo mi guía de valor inestimable a lo largo de todos estos años, quien a pesar de mis retrasos ha permanecido alentándome hasta ver finalizado este trabajo.

Del departamento de Matemática Aplicada, en el seno del cual se ha desarrollado esta memoria, he sentido el apoyo continuo de todos sus miembros. Gracias a todos y gracias muy especiales al profesor Alberto Pérez de Vargas, por su ayuda y el interés demostrado durante la elaboración de esta tesis, y a la directora del departamento, M<sup>a</sup> Cristina Martínez Calvo, por su consejo y sus valores humanos, que siempre han tenido particulares connotaciones de admiración y respeto. También debo dar las gracias a la profesora Carmen Fernández Chamizo por las valiosas sugerencias y correcciones aportadas en la realización de esta tesis.

Mi agradecimiento muy especial a Javier, profesor de la UNED, con quien tuve la suerte de ‘tropezar’ en uno de esos momentos en que el desánimo se apodera de ti y de buena gana uno tiraría la toalla. Pero de Javier obtuve una ayuda desinteresada que nunca podré olvidar. Gracias por las clases

magistrales que me ofreció desinteresadamente, gracias por su dedicación y por su apoyo incondicional.

La realización de un trabajo de investigación precisa de una gran cantidad de información puntual y actualizada. Esta labor le ha correspondido en su mayoría a la hemeroteca de la Facultad de Informática de la Politécnica, a ellos va mi más sincero agradecimiento por la destacada profesionalidad de todos sus componentes y muy especialmente gracias a Begoña y a Alberto. A Begoña, porque no hay búsqueda bibliográfica que se le resista y a Alberto, por no limitarse simplemente a atender a los alumnos por las tardes sino por estar pendiente de lo que he necesitado para llevar a cabo mi investigación. Gracias a todos por saber entregarme en el menor tiempo posible el material, que día a día he ido solicitando, y con la amabilidad que destaca a alguien que da sentido a su trabajo más allá de lo profesional. No quiero olvidar a M<sup>a</sup> Luisa, que además de amiga, es, junto con Alfonso, profesional de la biblioteca de Farmacia, a ellos he recurrido en momentos límite buscando esa referencia que, próxima ya la terminación de la tesis siempre falta, obteniendo en un tiempo récord la información solicitada.

No puedo olvidar a Miguel, ‘compañero de fatigas’ desde hace muchos años, acabamos juntos la carrera y juntos decidimos meternos en el departamento de Matemática Aplicada. Hemos compartido desesperaciones y alegrías, muchas reuniones ‘fuera de horario’ y muchas conversaciones por ‘mail’. Gracias Miguel por tu acompañamiento continuo y por tu persona. Espero que a ti también te llegue pronto este momento, hasta entonces, cuenta conmigo.

Un ofrecimiento y entrega fundamental he encontrado siempre en Concepción Gorostiza, responsable de informática del Ciemat, centro del cual formo parte, quien ha sabido darme en todo momento tanto un apoyo profesional como personal. En el centro de cálculo que ella dirige he podido disponer de cuantos recursos he necesitado para llevar a cabo la elaboración de esta tesis.

Gracias muy especiales al ilustre profesor D. Manuel, mi padre, maestro de vocación, maestro ‘de los de antes’, quien ha seguido paso a paso la elaboración de este trabajo. Sin su colaboración técnica el anexo no habría podido culminarse ya que la mayoría de los diagramas expuestos han sido ‘cosa suya’. La verdad es que es una suerte contar con un profesor en la familia, y yo la he tenido. Gracias D. Manuel, por haberte hecho un experto, de la informática que necesitabas para ‘construir los cuadros’, como tú decías. A tus setenta y tantos años de edad el ‘FlowCharter’, el ‘Visio Drawing’ y algún que otro programas más ya no tienen secretos para tí. Gracias por el esfuerzo que has hecho para que este trabajo viera el día de hoy y gracias por hacerlo siempre con el entusiasmo y la sobriedad en el trabajo ‘bien hecho’ que os caracteriza a los castellanos viejos.

A todos mis amigos, de quienes he obtenido siempre un apoyo casi heroico a pesar de mi encierro final, va mi agradecimiento más profundo. Siento no poder citarlos a todos personalmente porque sería interminable este capítulo, pero todos sabéis los nombres y estoy segura de que generosamente, como siempre, me liberáis de esta obligación.

Mi agradecimiento y admiración a mi hermana mayor, Maryví, farmacéutica y bióloga, con quien he tenido la suerte de compartir y comentar el desarrollo de este trabajo. Muchas de las ideas que se han tomado forma en esta memoria han surgido de conversaciones informales con ella. Gracias por tu crítica siempre acertada y tu constante apoyo.

Gracias mil a Emilio, por su incansable apoyo personal y científico. Con él he podido comentar, discutir y contrastar muchas de las ideas que poco a poco han ido llenando estas páginas. Tu papel ha sido difícil, al tener que sobrellevar esta tesis como carabina permanente durante todo el noviazgo y los primeros meses de casados. Cuando me conociste ella ya estaba ‘por medio’, y has afrontado heroicamente mis largas tardes frente al ordenador. A tí te quiero dar la enhorabuena porque, parece que después de los agobios de los últimos meses, esta tesis ya ha pasado a formar parte de nuestra historia. Gracias por tu comprensión y tus valiosos puntos de vista en muchos momentos clave y sobre todo por tu saber estar cuando te he necesitado. Sabes que una gran parte de ti va en estos dos tomos.

Intencionadamente he dejado para el último lugar a mis padres, a quienes les debo lo que soy. De ellos he aprendido a ser persona, porque me lo han enseñado con su ejemplo constante, y de ellos he recibido el continuo empuje para crecer y desarrollarme en todos los ámbitos de la vida. Gracias a mi madre, quien con su gran capacidad de sacrificio se ha preocupado de liberarme de todas las tareas posibles con el fin de que pudiera sacar mayor cantidad de tiempo para trabajar en la tesis. Gracias a mi padre, quien me ha alentado siempre en todo nuevo proyecto que he querido emprender. Ellos son los responsables de que esta tesis haya encontrado un final feliz. Gracias con todo mi cariño.

## **LISTA DE FIGURAS**

---

<b>Figura</b>	<b>Nombre</b>	<b>Pág.</b>
2.1.	Partes de un S.B.C.....	13
2.2.	Fases en la construcción de un S.B.C.....	14
3.1	Evolución de lenguajes de programación.....	37
3.2.	Tarjeta de clase en CRC/RDD.....	44
4.1.	Dogma central de la biología molecular.....	63
4.2.	Proceso de la biosíntesis de proteínas .....	64
4.3.	Vista del proceso de traducción de la biosíntesis de proteínas .....	64
4.4.	Niveles de la estructura proteica .....	65
4.5.	Comparación de la biosíntesis de proteínas con ALBA.....	69
4.6.	Etapas de ALBA .....	72
4.7.	Notación del modelo de síntesis de objetos visuales.....	73
4.8.	Notación del modelo de regulación de objetos visuales.....	74
4.9.	Notación del refinamiento del modelo de síntesis de objetos visuales.....	74
4.10.	Notación del refinamiento del modelo de regulación de objetos visuales.....	75
4.11.	Interacción Responsabilidad - Colaboración en ALBA .....	75
5.1.	Fases y subfases del modelo de síntesis de objetos visuales.....	78
5.2.	Dogma central del modelo de síntesis de objetos visuales .....	79
5.3.	Dogma gráfico del modelo de síntesis de objetos visuales.....	80
5.4.	Equivalencia de información .....	81
5.5.	Comparación de las fases de almacenamiento biológica e informática .....	82
5.6.	Comparación de las fases de transporte biológica e informática .....	86
5.7.	Comparación de las fases funcional biológica e informática.....	88
5.8.	Comparación de las fases de activación biológica e informática.....	90
5.9.	Comparación de las fases de traducción biológica e informática .....	94
5.10.	Fases del modelo de regulación de objetos visuales .....	97
5.11.	Fases del refinamiento del modelo de síntesis de objetos visuales .....	101
5.12.	Fases I y II del refinamiento del modelo de síntesis de objetos visuales.....	103

5.13.	Fase III del refinamiento del modelo de síntesis de objetos visuales .....	104
5.14.	Fases del refinamiento del modelo de regulación de objetos visuales .....	107
5.15.	Vista gráfica del refinamiento del modelo de regulación de objetos visuales .....	108
6.1.	Patrón de arquitectura de un generador de S.B.C. (PARGEN).....	114
7.1.	Estructura interna del módulo de validación y verificación .....	124
7.2.	Funcionamiento del analizador probabilístico de probabilidad subjetiva.....	133
7.3.	Funcionamiento del analizador probabilístico de redes bayesianas .....	136
8.1	Arbol de decisión para una base de conocimiento basada en reglas de producción.....	138
8.2.	Arbol de decisión para una base de conocimiento basada en marcos de clasificación .....	139
8.3.	Niveles específicos de la red de conceptos.....	141
8.4.	Estructura interna de los árboles de decisión.....	146
8.5.	Arbol de decisión multinivel.....	146
8.6.	Arbol de decisión multiplano-multinivel.....	147
8.7.	Propagación en una red de conceptos basada en marcos.....	148
8.8.	Propagación en una red de conceptos basada en reglas de producción.....	149
8.9.	Elaboración de las medidas de suficiencia y necesidad.....	151
8.10.	Propagación de la probabilidad subjetiva .....	152
8.11.	Propagación de los factores de certeza.....	153
8.12.	Propagación de la red bayesiana .....	156
9.1.	Esquemas de fichas descriptivas .....	160
9.2.	Esquemas de fichas conceptuales .....	161
9.3.	Esquemas de fichas funcionales .....	162
9.4.	Modelo de tres capas de la aplicación .....	164
10.1.	Diagrama de bloques de la aplicación .....	174
10.2.	Configuración general .....	176
10.3.	Representación/Presentación del conocimiento .....	176
10.4.	Propagación de la incertidumbre .....	176
10.5.	Menú 'Entorno' del subsistema de desarrollo .....	177
10.6.	Editor de Reglas .....	178
10.7.	Editor de Marcos causales .....	178
10.8.	Editor de Agrupaciones .....	179
10.9.	Editor de Proposiciones .....	180
10.10.	Editor de Conocimientos .....	180
10.11.	Editor de Probabilidad bayesiana .....	181
10.12.	Editor de Probabilidad subjetiva .....	181
10.13.	Opción 'Buscar' del Editor de Marcos causales .....	182
10.14.	Opción 'Imprimir' del Editor de Marcos causales .....	182
10.15.	Opción 'Probar' del subsistema de desarrollo .....	183
10.16.	Configuración Inicial (I) .....	184
10.17.	Configuración Inicial (II) .....	184
10.18.	Configuración Inicial (III) .....	184
10.19	Opción de menú 'Motor de Inferencia' del susbsistema de ejecución .....	184

## **LISTA DE TABLAS**

---

<b>Tabla</b>	<b>Nombre</b>	<b>Pág.</b>
3.1.	Elementos del Modelo de Objetos .....	47
3.2.	Elementos del Modelo Dinámico.....	48
3.3.	Elementos del Modelo Funciona .....	50
3.4.	Diagramas propios de UML .....	52
7.1.	Esquema de representación de un atributo .....	117
7.2.	Esquema de representación de un objeto .....	118
7.3.	Esquema de representación de una regla de producción .....	119
7.4.	Esquema de representación de un marco de clasificación .....	119
7.5.	Esquema de representación de un marco causal .....	120
8.1.	Niveles genéricos de la red de conceptos .....	140
8.2.	Estados de chequeo del motor de inferencia .....	143
9.1.	Equivalencia de elementos en la captura de necesidades .....	163
9.2.	Responsabilidad y Colaboración para el módulo de representación del conocimiento .....	165
9.3.	Responsabilidad y Colaboración para el MVVBC .....	166
9.4.	Fases de PARSOV para el caso de probabilidad subjetiva y redes bayesianas .....	168
9.5.	Responsabilidad y Colaboración para el caso de factores de certeza .....	168
9.6.	Responsabilidad y Colaboración del módulo de propagación de probabilidad subjetiva y redes bayesianas .....	170
9.7.	Responsabilidad y Colaboración del módulo de propagación de factores de certeza .....	171
10.1.	Pseudocódigo de la función ‘Asignar nodos’ .....	185
10.2.	Pseudocódigo de la función ‘Comprobar existencia de ciclos’ .....	186
10.3.	Pseudocódigo de la función ‘Recoger Camino’ .....	187



# INTRODUCCIÓN

**ASPECTOS GENERALES  
SISTEMAS BASADOS EN CONOCIMIENTO  
TECNOLOGÍA DE OBJETOS**

# **Capítulo 1**

---

## **Aspectos generales**

### **1.1. Motivación y objetivos**

Los sistemas basados en conocimiento (S.B.C.) son herramientas de consulta que tienen codificado el conocimiento y el razonamiento de un experto humano. La gran variedad de áreas de aplicación en que pueden emplearse dichos sistemas ha desencadenado el desarrollo masivo de herramientas genéricas para llevar a cabo su construcción, adoleciendo en algunos casos de cierta especialización cuando se trata de elaborar un S.B.C. concreto aplicable a una determinada materia. La Biología, como disciplina, proporciona respuestas a problemas de índole muy diferente, siendo por ello difícil encontrar un generador de S.B.C. que cuente con mecanismos de razonamiento adaptables a las distintas áreas que normalmente la Biología llega a manejar. Por esta razón se hace necesaria una herramienta informática que pueda tratar, tanto el conocimiento en que se basan los problemas de sistemática taxonómica, como el analizado en un problema de diagnóstico o bien el basado en

relaciones causales, para adecuar los mecanismos de comprobación con el fin de asegurar la coherencia del conocimiento recién introducido.

Por otro lado, hay cuestiones de las que se tiene un conocimiento no sólo cualitativo sino cuantitativo, por lo que al solicitar una consulta de las mismas, es preciso ofrecer tanto mecanismos de introducción de la información numérica como de propagación de la misma. Durante la elaboración de una consulta acerca de cualquier área relacionada con la Biología, siempre resultará más fácil concluirla si las preguntas formuladas se acompañan de recursos multimedia que proporcionen más información acerca del problema a resolver; por otra parte, conviene que la propagación del conocimiento obedezca a diferentes mecanismos según la información disponible.

Dadas todas estas premisas, nos parece oportuno plantear la construcción de un generador de S.B.C. que cubra los objetivos que detallamos a continuación:

1. Manejar diferentes tipos de representación del conocimiento, cualitativo y cuantitativo, asegurando la consistencia del mismo mediante mecanismos de comprobación.
2. Disponer de diferentes formas de presentación del conocimiento al usuario previas a la propagación del mismo.
3. Plantear las consultas al usuario empleando recursos multimedia.
4. Proporcionar mecanismos de propagación de la incertidumbre.

Actualmente el término de ingeniería de software parece que se ha sustituido por el de arquitectura de software, premiando la integración de diferentes módulos para la construcción de una herramienta informática frente a la aplicación de un conjunto de técnicas y metodologías que lleven a cabo las labores de análisis y diseño de una aplicación. Esta afirmación se traduce en que actualmente las metodologías de análisis y diseño se complementan con los patrones de diseño y con los patrones de arquitectura.

Los patrones de diseño se aplicaron por primera vez en el área informática por Cunningham [46]. Su objetivo es proporcionar una respuesta genérica en términos de diseño informático a un problema concreto, con el fin de que pueda aplicarse tantas veces como se quiera sin tener que analizar dicho problema en repetidas ocasiones. Los patrones de diseño basan su desarrollo en la experiencia del desarrollador. Cuando varios patrones de diseño se integran para desembocar en un desarrollo informático completo surgen los lenguajes de patrones.

Por otro lado, los patrones de arquitectura sirven para estructurar en componentes una determinada aplicación y para organizarlos entre sí con el fin de que se cumplan las especificaciones previas. De esta forma, diseño y arquitectura se complementan para llevar a cabo la construcción de una aplicación informática.

Como consecuencia de lo anteriormente expuesto, nos ha parecido adecuado proponer, por un lado, un lenguaje para la generación de patrones de diseño orientados a objetos, de forma que simplifique la complejidad del sistema a desarrollar por medio de la abstracción y favorezca su mantenimiento por medio de la reutilización; por otro lado, proponemos un patrón de arquitectura en el que se plasma la organización estructural en subsistemas y módulos para un generador de S.B.C. aplicable a áreas biológicas.

Dado el carácter marcadamente biológico de la herramienta a construir, hemos considerado interesante abordar la elaboración del lenguaje de patrones desde un punto de vista evolutivo. Así conseguimos que dicho lenguaje proporcione de manera intuitiva la resolución de las distintas fases del diseño de la aplicación mediante la creación de patrones de diseño, que hemos formalizado como modelos, basados en los mecanismos evolutivos de la propia naturaleza. En concreto la biosíntesis de proteínas ha servido como referencia fundamental para demostrar que la construcción de un sistema informático puede enfocarse desde un punto de vista evolutivo.

De esta forma y, siguiendo las pautas anteriormente expuestas, planteamos dos nuevos objetivos:

5. Desarrollar un lenguaje para la generación de patrones de diseño genéricos que favorezcan la creación a su vez de otros patrones de diseño específicos que ayuden a resolver problemas concretos. Este lenguaje debe proporcionar una notación clara e intuitiva, fácilmente transportable a un lenguaje visual y que conste de una serie de etapas deducibles a la vista del funcionamiento de los sistemas vivos.
6. Desarrollar un patrón de arquitectura orientado a objetos que sea aplicable a la construcción de generadores de S.B.C. de áreas biológicas.

En consecuencia creemos poder afirmar que esta tesis recorre todos los pasos necesarios para la construcción de una herramienta informática desde la elección de patrones de diseño adecuados, la elección del patrón de arquitectura que ayude a dilucidar los subsistemas y módulos de la herramienta la aplicación de dichos patrones a un caso concreto.

## 1.2. Presentación

El desarrollo de una herramienta informática puede resultar complejo tanto por la elección del lenguaje de programación, por la organización en subsistemas que se estime más adecuada, como por las técnicas de diseño más acordes con el sistema a desarrollar. De esta forma se impone estudiar detenidamente cada una de estas tres áreas con el fin de conseguir una integración perfecta entre lenguaje, arquitectura y diseño. Con esta base como premisa, en la elaboración de este trabajo

planteamos la confección de un **patrón de arquitectura** orientado a objetos que hemos denominado: ‘**Patrón para la ARquitectura de un GENerador de Sistemas Basados en el Conocimiento (PARGEN)**’ asignándole como objetivo la **construcción de un generador de sistemas basados en el conocimiento (S.B.C.) que específicamente pueda emplearse en áreas biológicas**. En términos generales este patrón propone una arquitectura modular, que se organiza en dos subsistemas con objetivos notablemente diferenciados: el primero, representa y comprueba la información introducida por el experto, y el segundo facilita la creación de consultas haciendo uso de mecanismos de inferencia. Asimismo, en cada uno de estos subsistemas se ha detectado una funcionalidad diversa que ha sido necesario integrar en módulos con el fin de llevar a cabo la construcción de la citada herramienta. El primer subsistema se ha convenido dividir en tres módulos: dos de ellos encargados, respectivamente, de representar la información cualitativa y cuantitativa, mientras que el tercero deberá comprobar su consistencia. Por otra parte, el segundo subsistema aparece integrado de igual forma por tres módulos, de manera que dos de ellos se hacen cargo de la propagación del conocimiento cualitativo y cuantitativo, respectivamente, mientras que al tercero se le asigna la presentación del mismo al usuario, justo antes de ser propagada.

El generador de S.B.C. propuesto admite tres formas de representación en sus aspectos cualitativo y cuantitativo. El conocimiento cualitativo puede expresarse en forma de marcos de clasificación (frames), reglas de producción y marcos causales. Los marcos causales son una forma particular de representación del conocimiento cuando se deseé construir una red bayesiana y constituyen una aportación del autor. El conocimiento cuantitativo recoge tres modos diferentes de tratamiento de la incertidumbre: la probabilidad subjetiva y los factores de certeza aplicables ambos únicamente a las reglas y las redes bayesianas, propias de los marcos causales.

Dados los continuos cambios que están sucediéndose en las técnicas de modelado orientado a objetos, se ha elaborado un **lenguaje para la generación de patrones de diseño** orientados a objetos basado en una arquitectura de tres capas. A este lenguaje lo hemos denominado: ‘**Lenguaje de Patrones para la Síntesis y Regulación de Objetos Visuales (ALBA)**’; su objetivo intrínseco es proporcionar las pautas necesarias para llevar a cabo el análisis y diseño de una aplicación informática. ALBA ofrece cuatro **patrones de diseño** genéricos, que se corresponden con cuatro modelos. Estos modelos van construyéndose a través de unas etapas que están inspiradas en los mecanismos evolutivos que emplea la naturaleza en su permanente evolución. De esta manera concretamos la formalización de este lenguaje buscando inspiración en las fases por las que atraviesa todo proceso biológico, habiendo centrado nuestro estudio en los mecanismos de expresión que representa la **biosíntesis de proteínas**.

La naturaleza ha evolucionado desde los organismos unicelulares, sin diferenciación organular, existentes hace millones de años, hasta los organismos pluricelulares marcadamente especializados por lo que, siguiendo esta observación científicamente probada y utilizando asimismo idéntica o similar

estructura, creemos que el desarrollo de cualquier sistema informático puede enfocarse desde un punto de vista evolutivo.

El lenguaje para la generación de patrones que proponemos mantiene como objetivos, tanto la elaboración de los llamados objetos visuales, que son en definitiva los que permitirán la introducción y mantenimiento de la información por parte del usuario, como la extracción de las interacciones entre ellos, lo que resolverá la funcionalidad que se precise. Paralelamente, se puede afirmar que el objetivo intrínseco de la biosíntesis de proteínas reside en la elaboración de las proteínas que necesita el organismo de referencia, macromoléculas que estimamos equivalentes a los objetos visuales en ALBA pero, además, aquéllas atenderán las diferentes necesidades del organismo, siendo éstas también equivalentes a la funcionalidad en ALBA.

La implementación formal de cualquier problema real parte de una descripción general que, ascendiendo en el nivel de complejidad, llega a una clara diversificación de funciones y, como consecuencia, a un alto grado de especialización. Dicha especialización queda plasmada en la creación de objetos entrenados en tareas concretas que forman parte del todo perdiendo por completo cualquier tipo de autonomía y precisando del resto del sistema para llevar a cabo la funcionalidad para la que fueron creados.

ALBA es un lenguaje para patrones que está basado en el patrón evolutivo que proporciona el mecanismo biológico de la biosíntesis de proteínas, por lo que sugiere la separación en tres capas del problema informático a implementar. Estas tres capas son las siguientes:

- a) 1<sup>a</sup> Capa o lógica de datos.- Referente al almacenamiento de la información.
- b) 2<sup>a</sup> Capa o lógica de la aplicación - Contiene la gestión de la información.
- c) 3<sup>a</sup> Capa o Interfaz gráfica del usuario.- Agrupa todo el conjunto de pantallas que van a interaccionar directamente con el usuario.

Cada una de estas capas está constituida por una serie de objetos particulares cuya responsabilidad viene determinada por la capa a la que pertenece; así los objetos mensajeros forman parte de la primera capa, los objetos gestores de la segunda y los objetos visuales de la tercera. Los objetos mensajeros almacenan la información y su nombre se debe a que la función que ejercen guarda cierta similitud con el papel que juega el ARNm en el organismo. Los objetos gestores transportan la información de un sitio a otro y su misión viene a ser equivalente a la ejercida por el ARNt. Los objetos visuales permiten el manejo directo de la información por parte del usuario gracias a sus recursos gráficos y presentan, en esencia, una cierta equivalencia con las proteínas.

Una vez descritas las responsabilidades de cada uno de los objetos y las colaboraciones establecidas entre ellos, es necesario comprobar su correcto funcionamiento mediante la aplicación de los patrones de diseño propuestos a un caso concreto. Dicho caso concreto consiste en la construcción

de un generador de S.B.C. para áreas biológicas, descrito en el ANEXO de esta tesis, en el que queda plasmado que tanto PARGEN como ALBA desembocan en un desarrollo completamente funcional.

Por último, queremos señalar que tanto la estructura como el propio desarrollo de este trabajo, especialmente en la parte que muestra la forma en que el lenguaje para patrones de diseño utiliza como molde el patrón evolutivo que proporciona la biosíntesis de proteínas, se ha considerado conveniente ilustrar con figuras. Estas figuras ayudan a entender el evidente paralelismo funcional que existe entre ambos, advirtiendo, además, que dichas figuras son originales en cuanto a su composición pero algunos de sus elementos se han extraído de libros o documentos electrónicos, ya que lo que queremos transmitir es la comparativa propiamente dicha y no detenernos demasiado en la originalidad de algunas partes de las mismas, como pueden ser los fragmentos de ADN o los ribosomas, empeño que, además de su dificultad en la confección, no aportaría ningún valor nuevo a la presente memoria.

Como nota final, debemos destacar que el cuerpo general de la tesis se recoge en un único TOMO, presentando en ANEXO (tomo II) los resultados de integrar ambos patrones: ALBA y PARGEN, que han dado como resultado el desarrollo de un generador de S.B.C. para áreas biológicas. Podemos afirmar que esta sistemática nos ha parecido de más fácil comprensión y más útil y manejable en su consulta.

### 1.3. Organización

La presente tesis aparece dividida en cinco partes, cada una de las cuales está formada por capítulos, agrupados según un mismo objetivo.

**La parte I** está formada por tres capítulos que constituyen una **introducción teórica** a la presente memoria de forma que recoge la presentación de la tesis que, a su vez, va precedida de explicaciones sobre la motivación que ha inducido a su desarrollo y los objetivos que se pretenden. A continuación se incluye una breve descripción de cómo se han organizado las partes de la misma. Los capítulos siguientes sitúan al lector en las dos áreas que va a manejar este trabajo: los Sistemas Basados en el Conocimiento y la Tecnología de Objetos. En cuanto a los primeros, se describen algunas de las áreas de que consta la construcción de S.B.C, en concreto aquéllas que guardan cierta relación con la elaboración de esta memoria y en segundo lugar se describe la Tecnología de Objetos, desde sus aspectos básicos hasta la construcción de patrones, detallando los distintos pasos evolutivos que se han ido sucediendo. Estos dos capítulos se centran fundamentalmente en los temas que van a ser tratados y ampliados posteriormente.

**La parte II, elaboración de un lenguaje de patrones,** consta de dos capítulos. El primero de ellos consiste en una breve exposición del mecanismo biológico en que se basa ALBA: la biosíntesis de proteínas. El segundo constituye la elaboración del lenguaje para la generación de patrones de

diseño, ALBA, aplicable a cualquier problema real que requiera la visualización de una información previamente almacenada. En este segundo capítulo especificaremos los cuatro patrones de diseño genéricos a que da lugar dicho lenguaje. Estos cuatro patrones presentan como objetivo común la elaboración del análisis y diseño de un sistema informático. Por otro lado se detallan algunos patrones de diseño específicos que han surgido a partir de los patrones de diseño genéricos y que van a aplicarse en la construcción de generadores de S.B.C.

**La parte III, elaboración de un patrón de arquitectura,** incluye la elaboración del patrón de arquitectura PARGEN cuyo objetivo es la construcción de un generador de S.B.C. Dicho patrón propone la creación de dos subsistemas: uno primero de desarrollo que representa la introducción de la información cualitativa y cuantitativa, y un segundo subsistema de ejecución que concierne a la propagación tanto de los conocimientos como de la incertidumbre. Cada uno de estos subsistemas se corresponde con un capítulo.

**La parte IV, desarrollo informático,** engloba la aplicación de las aportaciones anteriormente expuestas a un caso práctico concreto. En esta parte se plasma el resultado de integrar los patrones de diseño genéricos con el patrón de arquitectura. El primer capítulo aporta una técnica para formalizar los requisitos antes de llevar a cabo el diseño de cualquier tipo de herramienta y que es propia de la autora. A continuación, los siguientes apartados del primer capítulo detallan cómo dependiendo del tipo de subsistema y módulo de PARGEN, se aplican de forma diferente unos patrones de diseño u otros. Este capítulo sirve como introducción teórica al caso práctico desarrollado en el anexo. En el anexo se exponen los resultados de integrar ALBA con PARGEN, poniéndose de manifiesto que dicha integración desemboca en un desarrollo práctico completamente funcional. De esta forma hemos considerado oportuno comentar en este capítulo los resultados plasmados en dicho anexo. El tercer capítulo de esta parte muestra los detalles de la implementación informática cuando se utiliza un lenguaje concreto de programación. En este caso hemos empleado Visual C++ 6.0 y su librería de clases MFC 4.0.

**La parte V enumera las conclusiones** a las que ha llegado esta memoria, las aportaciones realizadas por la autora así como algunas líneas para futuras investigaciones que podrían programarse como continuación de este trabajo.

Por último, aparte de la bibliografía que se cita en apoyo a nuestra tesis y que supone la base científica de todo el trabajo, se destaca la presentación de un ANEXO (tomo II) cuya organización es similar a la del tomo I. El ANEXO se organiza en tres partes y cada una de estas partes está formada por una serie de capítulos.

**La parte I, generador de sistemas basados en conocimiento,** consta de un único capítulo en el que se describe el conjunto de requisitos que ha de cumplir el generador de S.B.C. a desarrollar.

**La parte II, subsistema de desarrollo,** incluye tres capítulos, que se corresponden con cada uno de los módulos de que consta dicho subsistema en el patrón de arquitectura propuesto. Para cada uno de dicho módulos se han desarrollado los cuatro patrones de diseño que constituyen el lenguaje de patrones ALBA.

**La parte III, subsistema de ejecución,** incluye tres capítulos, que se corresponden igualmente con los módulos en que se subdivide dicho subsistema. De la misma forma que la parte anterior, cada módulo desarrolla de manera particular los patrones constituyentes de ALBA.

**La parte IV, apéndices,** describe la notación empleada para cada una de las fases de ALBA, así como los códigos de error empleados para cada operación.

**La parte V, índices de figuras,** representa todas las figuras existentes en el anexo, junto con la página donde se encuentran.

## **Capítulo 2**

---

### **Sistemas Basados en Conocimiento**

#### **2.1. Fundamentos generales**

Los Sistemas Expertos (S.E) constituyen una rama de la Inteligencia Artificial (I.A.) cuyos orígenes se remontan a 1950, cuando Alan Turing [171], propuso el test denominado por él mismo "the Imitation Game" con el fin de determinar si una máquina puede considerarse inteligente. Este test consiste en que la máquina debe ser capaz de mantener una conversación como si fuera un ser humano. El término de I.A. fue acuñado en 1955 por John McCarthy, profesor de la Universidad de Stanford. Su área principal de investigación era la formalización del conocimiento del sentido común [116]. El desarrollo de la I.A. ha dado lugar al desarrollo de otras áreas. Por su relación con lo desarrollado en esta tesis, hablaremos en concreto de dos de ellas: la programación simbólica y el razonamiento aproximado.

La programación simbólica sugiere el nacimiento de lenguajes como LISP (LISt Processor), inventado por McCarthy en 1958, basado en listas y símbolos, y que admite funcionalidad recursiva [117,118]. LISP es el primer lenguaje de programación que se asocia con más fuerza a la I.A.. Se trata de un lenguaje funcional con ciertas extensiones procedurales.

Más adelante, basándose en las ideas sobre programación lógica de Kowalski, Colmerauer y Roussel crean en 1973 el lenguaje PROLOG [32] que implementa la lógica de predicados aunque con ciertas limitaciones.

La I.A. también contribuye al desarrollo de lenguajes Orientados a objetos, tales como SMALLTALK y Objective C, extensiones de lenguajes que presentan la particularidad de ser orientados a objetos, como CLOS (Common LISP Object System) [16], extensión de LISP, L&O (Logic & Objects), extensión de PROLOG o C++, extensión de C.

En cuanto al razonamiento aproximado, trata del manejo de la incertidumbre que surge cuando se desarrolla cualquier tipo de aplicación a problemas del mundo real. Normalmente puede existir ignorancia acerca de una determinada información, imprecisiones, errores o conocimiento incompleto. Por esta razón se han desarrollado una serie de teorías y aproximaciones de las que hablaremos más adelante.

Una de las primeras definiciones que aparecen para un S.E. es proporcionada por Feigenbaum durante la celebración del congreso mundial de I.A. Feigenbaum considera un S.E. como un “*programa de computador inteligente*” cuyo objetivo esencial es dar soluciones a problemas que por ser lo suficientemente difíciles, requieren la intervención de expertos humanos, haciendo uso de mecanismos de inferencia para ofrecer una solución [47]. Posteriormente, a medida que la tecnología ha ido avanzando, la definición de S.E. ha ido tomando otras connotaciones, de forma que ya Castillo habla de un S.E. como un “...*sistema informático...*” [25]. Gutiérrez, por otra parte, considera un S.E. como un “...*sistema integrado...*”, [73] de forma que el S.E. ya no es un simple programa sino un sistema formado por la integración de varios componentes, cada uno con una funcionalidad diferente, tema del que hablaremos más adelante. Para más información sobre las generalidades de los SS.EE y sus orígenes, se puede consultar Liebowitz [110] y Waterman [181].

DENDRAL surge como pionero de los SS.EE. Fue desarrollado por Lederberg, Feigenbaum y Buchanan en 1965. DENDRAL era un programa capaz de deducir la estructura de moléculas complejas, a partir de su espectograma de masas. DENDRAL introdujo el uso de reglas para la representación del conocimiento y razonamiento simbólico a través de conceptos, separaba conocimiento de inferencia. A medida que DENDRAL fue evolucionando [20,21] y, dado su éxito, Feigenbaum, Buchanan y Shortliffe se aventuraron en el desarrollo de un S.E. para diagnóstico médico, denominado MYCIN [161].

MYCIN, en contraste con DENDRAL, precisaba del diálogo continuo con el experto. MYCIN empleaba reglas como mecanismo de representación del conocimiento. Presentaba un mecanismo matemático de representación y propagación de la incertidumbre basado en factores de certeza, bastante acorde con la forma que tienen los médicos de evaluar el impacto de los síntomas; su motor de inferencia presentaba encadenamiento de las reglas hacia atrás. MYCIN fue el primer S.E. capaz de explicar su razonamiento. Separaba por un lado las reglas y por otro el motor de inferencia, de ahí que

como consecuencia, surgiera EMYCIN [173], un motor de inferencia basado en MYCIN. MYCIN fue desarrollado en LISP. Una mejora de MYCIN fue TEIRESIAS [48,49], que introdujo el uso de meta-reglas en la base de conocimiento.

Posteriormente apareció PROSPECTOR [55] como S.E. de razonamiento probabilístico, cuyo objetivo era aconsejar perforaciones geológicas concretas en determinados emplazamientos. El conocimiento lo almacenaba en forma de reglas y la propagación del conocimiento se realizaba mediante técnicas bayesianas aproximadas. PROSPECTOR fue el primer éxito comercial dentro del área de los SS.EE.

CASNET [102, 183] surgió en los años 70 para el diagnóstico del glaucoma. El conocimiento lo representaba mediante una red causal y se propagaba mediante técnicas heurísticas. Su motor de inferencia dio lugar al desarrollo por separado a una herramienta denominada EXPERT [103].

PIP (Present Illness Program) fue el primer S.E. basado en marcos; se desarrolló en la década de los 70 y se aplicó también en el campo de la medicina [132,169]. PIP presentó como ventajas por un lado el proporcionar como resultado varios diagnósticos posibles y por otro el presentar una cierta flexibilidad en el momento de hacer las preguntas al usuario, frente a la rigidez de los sistemas tradicionales que basaban las consultas en árboles de decisión.

Durante los ochenta, con la introducción de los ordenadores personales, se produjo un desarrollo masivo de generadores de S.E. [25], fáciles de usar, con lo cual comienzan a proliferar S.E. desarrollados por estudiosos de todas las ciencias; cobran importancia posibles áreas de aplicación de los mismos como son la clasificación, predicción, diagnosis, interpretación, diseño y validación. A pesar de todo, el área predominante de desarrollo ha sido, en su mayor parte, el diagnóstico. [56]. Si se desea conocer la proporción de S.E. desarrollados en diferentes áreas de aplicación, Durkin plantea un estudio realizado sobre 2500 S.E. y en Castillo se puede ver una adaptación del estudio de Durkin, observándose claramente las áreas de más aplicación. [26]. Es en esta época cuando empieza a aflorar una comercialización de los mismos. A finales de los ochenta, con el desarrollo de la tecnología de orientación a objetos, los SS.EE. evolucionan hacia los Sistemas Basados en Conocimiento (S.B.C.).

Los S.B.C. llevan a cabo las tareas para las que han sido creados mediante la representación de su conocimiento y mediante el razonamiento. Igual que la definición de los S.B.C. ha ido evolucionando con el avance de las nuevas tecnologías, su estructura también ha ido desarrollándose, de forma que inicialmente se consideraban como esenciales las cuatro partes que se detallan a continuación y que se ilustran en la figura 2.1:

- **Base de Conocimiento (Knowledge Base)**.- Conjunto formalizado de todos los conocimientos introducidos. El objetivo de la base de conocimiento es almacenar estáticamente los conocimientos.

- **Base de hechos (Working Memory).**- Conjunto formado por los resultados de la inferencia que va haciendo el sistema según se van contestando preguntas por el usuario. El objetivo de la base de hechos es almacenar dinámicamente los conocimientos, que varían en función de la consulta realizada.
- **Motor de Inferencia (Inference Engine).**- Conjunto de algoritmos que combinan la base de conocimiento con las respuestas proporcionadas por el usuario. Su objetivo es proporcionar soluciones durante el desarrollo de una consulta con el usuario.
- **Interfaz de usuario.**- Trata de hacer más sencilla y eficaz la elaboración de las consultas. Su objetivo es hacer las preguntas pertinentes al usuario y recoger las respuestas para que el motor de inferencia siga la base de conocimiento buscando una conclusión.

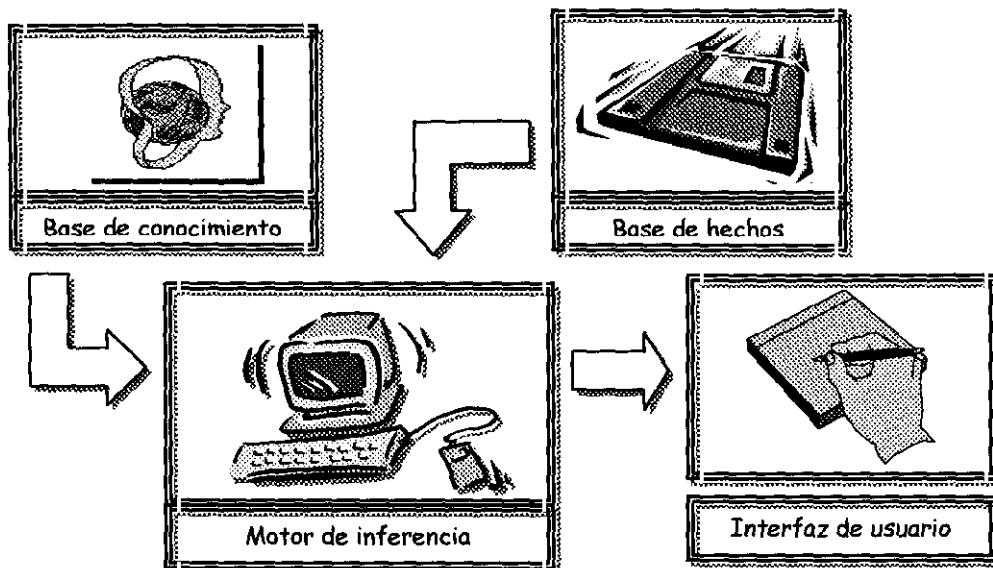


Figura 2.1.- Partes de un S.B.C.

Además de los componentes expuestos, posteriormente se han ido desarrollando otros que se detallan a continuación, cuya misión fundamental es mantener los componentes anteriormente citados:

- **Editor del conocimiento.**- Se encarga de recibir el conocimiento que formará parte de la base de conocimiento y de comprobar su consistencia mediante el componente para el control de la coherencia, que se encuentra incluido dentro de éste.
- **Base de datos.**- Consiste en el repositorio de información adicional que generalmente complementa a la base de conocimiento.

- Componente de explicación.- Proporciona una justificación de por qué el sistema hace ciertas preguntas o de cómo ha alcanzado las conclusiones pertinentes.
- Componente de aprendizaje.- Consiste en el incremento del conocimiento adquirido o en la deducción de mecanismos que mejoren la eficiencia del S.B.C.

El desarrollo de un S.B.C. viene determinado por una serie de fases cuyos objetivos son, en definitiva, actuar sobre cada uno de los componentes anteriormente expuestos. Estas fases son las que se enuncian a continuación, ilustrándose en la figura 2.2:

- a) Adquisición del conocimiento
- b) Representación del conocimiento
- c) Control de la coherencia
- d) Propagación

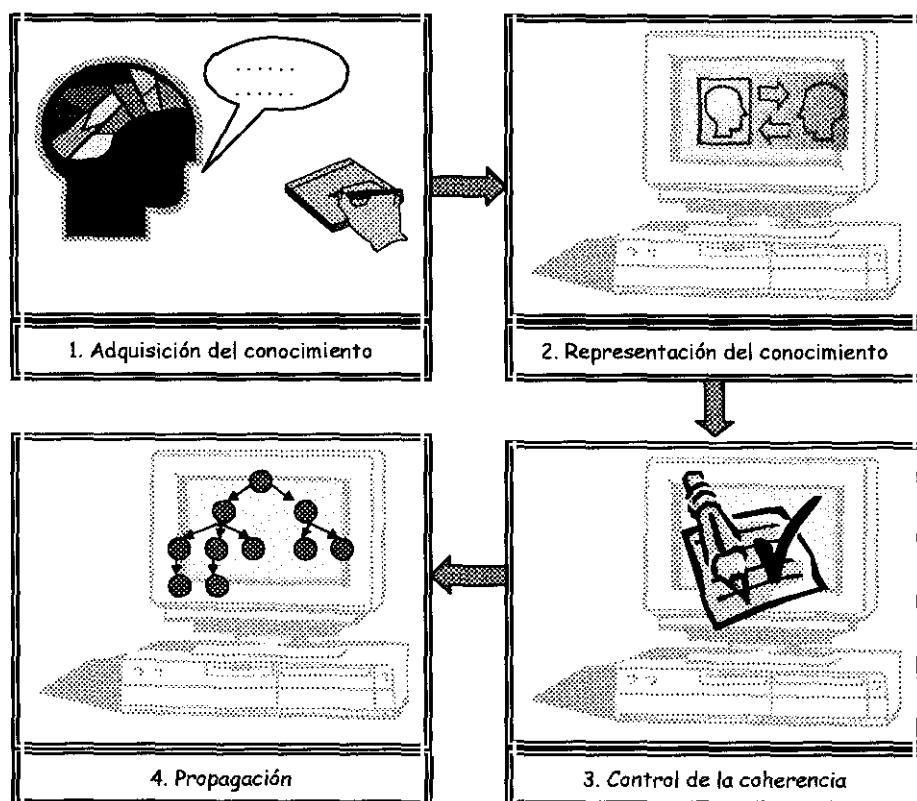


Figura 2.2.- Fases en la construcción de un S.B.C.

Los S.B.C. presentan una serie de características que les distinguen de otro tipo de sistemas informáticos, tales como las enunciadas a continuación:

- Son especializados, puesto que cada uno se aplica a un área de conocimiento concreta.
- Generalmente presentan un razonamiento heurístico, es decir, basado en la experiencia.
- Separan conocimiento e inferencia.
- Explican su razonamiento, por lo general.
- Poseen capacidad de diálogo, la mayor parte de las veces.

El desarrollo de los S.B.C. ha traído consigo un gran número de ventajas, entre ellas podemos destacar:

- Permanencia.- En el caso de un experto humano, el conocimiento desaparece con el experto o bien éste puede olvidar el conocimiento adquirido, de cualquier forma, un S.B.C. asegura que el conocimiento permanece siempre independientemente de la persona que lo hubiera adquirido.
- Multiplicidad.- Se puede hacer uso múltiple de un determinado conocimiento experto, no hace falta la presencia del mismo experto humano para resolver un problema.
- Eficacia.- Un S.B.C. no está afectado por causas externas, tales como el cansancio o la salud. Un S.E. es eficaz siempre al mismo nivel, al carecer de componentes humanos, salvo avería.
- Rapidez.- Un S.B.C. presenta generalmente un tiempo de respuesta mínimo.
- Coste.- Un S.B.C. puede llegar a amortizarse rápidamente si su funcionamiento responde a lo que se esperaba. Su coste de desarrollo suele ser elevado, pero dada la enorme utilidad de los mismos, a la larga tiene un bajo coste.
- Documentación.- Un S.B.C. puede dar una respuesta por escrito, en la cual se explica su proceso de razonamiento, es decir, cómo ha llegado a una determinada conclusión.

Sin embargo, los S.B.C. también tienen una serie de desventajas, tales como las que citamos a continuación:

- Intuición: Un S.B.C. carece de intuición y de sentido común; a pesar de los esfuerzos llevados a cabo por McCarthy, entre otros, no se ha llegado a una solución acerca de este tema.

- No resuelven un problema que esté fuerza del área para el cual se creó.
- Generalmente no conocen sus propios límites.
- La interacción que presentan es limitada puesto que carecen de sentidos: no ven, no oyen.

El desarrollo de los S.B.C. concretos ha proliferado en los últimos años, pero la construcción de herramientas generadoras de S.B.C. no se ha quedado atrás. Se han empleado diversidad de lenguajes para la programación de dichas herramientas. A continuación enunciamos ejemplo de generadores programados en tres tipos de lenguajes diferentes.

CLIPS [146], desarrollado por la NASA, es un generador de S.B.C. que incorpora programación estructurada, programación basada en reglas y orientación a objetos. Genera bases de conocimiento cuya unidad de conocimiento está formada por reglas de producción y objetos. Emplea como lenguaje de programación el lenguaje C.

EXPERT TALK es un generador de S.B.C. que también emplea como representación del conocimiento las reglas y objetos. Proporciona manejo de lógica difusa. El motor de inferencia presenta encadenamiento hacia delante y hacia atrás. Desarrollado por Nicola A.G. Zordan. Emplea como lenguaje de programación SMALLTALK.

BABYLON [30] es un entorno de desarrollo híbrido para S.B.C. con objetos y reglas como forma de representación del conocimiento. El motor de inferencia presenta encadenamiento hacia delante y hacia atrás. Está desarrollado en LISP.

FLESH [69] admite conocimiento en forma de proposiciones que se pueden propagar hacia delante y hacia atrás; y en forma de predicados, permitiendo la propagación hacia atrás en este caso. Admite el uso de lógica difusa.

R1/XCON fue un S.E. del que derivó OPS5, basado en reglas y con encadenamiento hacia adelante. Otros generadores dignos de mencionar son ART y KEE, cuya forma de representación del razonamiento viene dada por un mecanismo híbrido de marcos y reglas. Ambos se diseñaron para máquinas LISP y presentan como novedad notable su avanzada la interfaz gráfica.

GOLDWORKS y NEXPERT, cuyas ventajas fundamentales residen en su independencia de plataforma y en su capacidad gráfica, simplificando sobremanera la elaboración de un S.E. por muy complejo que éste sea.

## 2.2. Adquisición del conocimiento

La clave de la adquisición del conocimiento, es encontrar los conceptos sobre los que se construye el conocimiento y las relaciones existentes entre ellos [38]. Aunque hemos de afirmar que hoy en día se insiste más en el modelado del conocimiento, más cerca del razonamiento, que en la propia adquisición del mismo. Lo que sí es cierto es que el principal potencial de un S.B.C. se basa en el conocimiento que maneja, pudiéndonos plantear las preguntas:

- ¿Cómo automatizar la adquisición del conocimiento?
- ¿Cómo automatizar la calidad del conocimiento adquirido?

Históricamente la adquisición del conocimiento se ha venido llevando a cabo por un ingeniero del conocimiento, cuyas tareas a realizar eran las siguientes:

- Entrevistar al experto, aunque no siempre.
- Identificar las componentes del conocimiento.
- Construir el sistema experto.

La adquisición del conocimiento consiste en la construcción de un modelo conceptual que refleje tanto el conocimiento del que el experto dispone como su forma de razonar. La fuente de conocimiento puede ser una fuente directa (el propio experto) o indirecta (datos, ejemplo concreto, documentación en general). Algunas de las formas de adquirir el conocimiento son:

- Mediante entrevistas;
- Mediante aprendizaje por interacción (el ordenador ayuda a extraer el conocimiento) y
- Mediante aprendizaje por inducción, en el cual el propio ordenador, a partir de documentación basada en ejemplos, extrae el conocimiento.

La forma más empleada ha sido la de las entrevistas, pero como todo proceso en que interviene la capacidad humana, es la más susceptible de errores.

La adquisición del conocimiento comenzó su andadura llevando a cabo entrevistas con experto, haciendo una primera formalización en papel de la información extraída, mediante distintos tipos de recursos, como por ejemplo las tablas de conocimiento [138]. Posteriormente comenzaron a desarrollarse metodologías para hacer más fácil la adquisición del conocimiento. KOD (Knowledge Oriented Design) es una metodología desarrollada por científicos e ingenieros de software para

extraer, representar y verificar el conocimiento experto de cualquier dominio [104]. KADS es un modelo propuesto de tres capas que obtiene, interpreta y formaliza el conocimiento experto con el fin de que el S.B.C. pueda usarlo [185].

Según iba avanzando la década de los noventa, parece necesario hacer más hincapié en el mantenimiento de los S.B.C. más que en el propio desarrollo. La elevada cantidad de S.B.C. plantea la obligación de proponer mecanismos óptimos de mantenimiento que pueda realizar el propio experto, sin necesidad de recurrir al ingeniero del conocimiento. Así surgen una gran variedad de módulos de adquisición del conocimiento integrados dentro de los mismos generadores de S.B.C.

KNOWBEL [126] pide al usuario el conocimiento en forma de objetos y relaciones e internamente, mediante generalización, clasificación, agregación y relaciones temporales, lo transforma en unidades proposicionales; OOEKA [177] admite el conocimiento de forma gráfica, mediante una simbología, elaborada por él mismo y lo prepara para la construcción de la base de conocimiento, que puede ser reutilizada por otros S.B.C. La novedad de OOEKA es que propone la separación entre conocimiento y representación. Wu [190] desarrolla una técnica para facilitar la adquisición del conocimiento mediante la creación de ontologías [66].

Algunos ejemplos del empleo de técnicas de adquisición del conocimiento son [68, 57], que hacen uso de KOD, o bien [6] y [7] que emplean tablas de conocimiento y árboles de decisión, respectivamente.

## 2.3. Representación del conocimiento y razonamiento

La representación del conocimiento consiste en la codificación del conocimiento aportado por el experto con el fin de que forme parte de la base de conocimiento de un S.B.C. y puedan llevarse a cabo tareas de razonamiento. La representación del conocimiento lleva consigo previamente la adquisición del mismo por parte del experto o grupo de expertos. Durante el desarrollo de los S.B.C. han ido surgiendo una gran variedad de formas de representar el conocimiento. A continuación vamos a ir describiendo las más comunes.

- **LÓGICA**.- La lógica de predicados de primer orden o lógica simbólica constituye un mecanismo sencillo de representación del conocimiento, en el cual es fácil representar hechos, pero hay algunas desventajas como la incapacidad para representar la incertidumbre. Existe otro tipo de lógicas, como la lógica modal, aplicada en la representación del conocimiento por [122] o la lógica difusa [200, 201].
- **REDES ASOCIATIVAS**.- Consisten en un conjunto de nodos y aristas, donde los nodos representan entidades y los enlaces, las relaciones entre ellas. Entre ellas destacan las redes

semánticas [143] surgen como estructuras destinadas a la traducción de conceptos de un lenguaje escrito a otro. Específicamente se pretendía con ellas representar el significado de las palabras en inglés, basándose en las relaciones del tipo: 'Is A'. Las redes semánticas son estructuras declarativas, puesto que están formadas por nodos que se emplean para describir conceptos. Otro tipo particular de redes asociativas son las redes causales, que representan un modelo en que los nodos representan variables y los enlaces relaciones de causalidad. .

- **REGLAS** [128].- Constituyen la forma de representación más habitual en los S.B.C. PROSPECTOR fue el primer S.B.C. que las empleó para construir la base de conocimiento. MYCIN empleó las reglas tal y como las concebimos hoy en día. Su estructura esencial está formada por un antecedente, formado a su vez por un conjunto de premisas que se han de cumplir, y el consecuente, que agrupa el conjunto de conclusiones que se deducen de las premisas. El razonamiento de un S.B.C. con reglas como forma de representación se basa en un encadenamiento hacia delante o basado en datos, o bien en un encadenamiento hacia atrás, o basado en objetivos.
- **MARCOS** [121].- Según Minsky cuando nos encontramos ya sea ante una situación nueva, o ante la variación de una ya existente, selecciona de la memoria una estructura, denominada marco. El marco es una estructura de datos cuyo objetivo es representar una situación estereotipada. Los marcos pueden organizarse en una red de nodos y relaciones. Los niveles superiores de los marcos están fijados y los inferiores están formados por una estructura terminal que representa las ranuras (slots), que se pueden llenar tanto por datos como por procedimientos; así, si comparamos los marcos con las redes semánticas, los marcos presentan una naturaleza procedural aparte de su dimensión declarativa, en contraste con las redes. Una aplicación concreta de los marcos puede consultarse en [29].

Los S.B.C. suelen tener las reglas de producción como denominador común, en sus técnicas de representación del conocimiento. Un número excesivo de reglas de producción puede generar una explosión combinatoria durante el proceso de la inferencia [70]. Una solución a este problema viene dada por la extracción de las metas a partir del conocimiento introducido. Cada meta se dispara por el encadenamiento secuencial de un conjunto de reglas, que se ha dado en llamar esquemas [28]. Así surgen herramientas como PATH HUNTER [27] que extraen los caminos de las reglas a partir de la especificación de una meta.

Un intento de proporcionar un enfoque orientado a objetos a la representación tradicional de las reglas de producción se ilustra en un trabajo publicado por Boufriche-Boufaïda, en 1998, donde se representan las reglas en forma de red [19]. A medida que los S.B.C. han ido evolucionando, como decíamos anteriormente, se ha detectado la necesidad de combinar diferentes tipos de representación del conocimiento en una sola [72]. Surgen de esta manera los sistemas híbridos que han alcanzado pleno desarrollo a mediados de la década actual. Los sistemas híbridos presentan grandes ventajas, por

ejemplo complementar el conocimiento heurístico de las reglas de producción con el conocimiento declarativo de los marcos [109].

MANTRA es un S.B.C. que reúne un total de cuatro formas diferentes de representación del conocimiento, que interactúan entre sí mediante algoritmos híbridos de inferencia, constituyendo lo que el autor ha denominado una semántica unificada [15].

Otra cuestión relacionada con la representación del conocimiento es la forma en que está almacenada la información que va a constituir la base de conocimiento. Así, si el conocimiento de que se dispone viene almacenado en bases de datos particulares en forma de ejemplos, se puede construir un sistema que lleve a cabo tareas de aprendizaje, mediante la extracción de una regla de clasificación a partir de dichos ejemplos [194]. Si por otro lado se cuenta con un conocimiento que proviene de documentación escrita en lenguaje natural, el paso de reglas de este tipo de conocimiento puede resultar un tanto tedioso, de ahí, que se proponga como solución, emplear un sistema de representación del conocimiento mixto, basado en lógica de Horn, que haga más sencilla e inmediata la transposición de la base de conocimiento a un sistema informático [82]. Una alternativa puede ser organizar en principio el conocimiento en forma tabular, tarea realizada por el ingeniero del conocimiento, y a continuación, llevar a cabo la construcción de marcos y reglas a partir del mismo, [123], de esta forma se separa perfectamente la ingeniería del conocimiento de su representación.

Cuando aumenta la capacidad y variedad de información, se han desarrollado herramientas para representar información multimedia como UMART [164]. Si una unidad de conocimiento, proporciona diferentes tipos de información según el punto de vista desde el cual se analice, se pueden emplear herramientas como VIEW RETRIEVER [1] que, a partir del nombre de un concepto determinado, muestra al usuario todo el conjunto de descripciones del objeto desde el punto de vista que éste desee. Se pueden construir bases de conocimiento capaces de responder a cualquier pregunta que se les plantea, mediante la creación dinámica de nuevas representaciones de conceptos [31]. Otra cuestión importante de los S.B.C. es el mantenimiento, que parece mucho menos costoso cuando se emplean para su desarrollo lenguajes orientados a objeto y bases de reglas modulares [109].

## 2.4. Control de la coherencia

Esta fase consta de una serie de algoritmos encargados de comprobar que la información introducida en el sistema es consistente, tanto la cualitativa (conocimiento tal cual) como la cuantitativa (referente a la incertidumbre). Una vez construida la base de conocimiento, se procede a comprobar su coherencia con el fin de que pueda propagarse el conocimiento de forma adecuada. Para llevar a cabo esta tarea es necesario realizar labores de prueba, refinamiento y mantenimiento. La elaboración de un módulo adicional que realice de manera automática procesos de validación y verificación favorece que sean los mismos expertos, en vez de los informáticos, los que realicen el

mantenimiento de las bases de conocimiento [205], de ahí que desde mediados de los 80 se haya producido una evolución de esta nueva disciplina de los S.B.C., tal y como especifica Nazareth en un artículo publicado en 1993 [127].

En primer lugar, podemos aclarar las diferencias que existen entre la validación y la verificación. La validación asegura que el comportamiento externo de un S.B.C., está de acuerdo con el estudio inicial de necesidades [140]. La verificación asegura que la base de conocimiento de un S.B.C. está libre de inconsistencias internas [140]. Se han desarrollado gran número de herramientas y programas que tienen como objetivo asegurar la consistencia de la base de conocimiento creada por los expertos. El primer programa que se desarrolló con este fin fue uno que verificaba que la base de conocimiento ONCOCIN, un S.B.C. para diagnóstico oncológico, estaba completa y era consistente [168]. A partir de este primer desarrollo son muchas las herramientas que surgen para comprobar una base de conocimiento basada en reglas. Fundamentalmente se dice que como las bases de reglas son tan sencillas de construir, casi siempre tienen un gran número de incorrecciones [105] teniendo que acudir a un módulo que verifique la consistencia de lo introducido.

Así por ejemplo destaca CHECK [129] que verifica la consistencia y completitud de una base de conocimiento; COVER [140] y DIVER [205] ambos aplicados a sistemas basados en conocimiento que emplean como técnica de representación la lógica de primer orden; IN-DEPTH II [119] aplicado a sistemas basados en conocimiento multinivel; PREPARE [204] está basado en una clase especial de redes de Petri denominada redes de Predicado/Transición. Kiper propone una validación gráfica a partir de la generación de los posibles caminos de las reglas con el fin de hacer más sencillo la posterior comprobación [99]. Murrell mediante la elaboración de tablas de decisión, y posteriormente la transformación en un grafo, lleva a cabo las labores de validación y verificación de reglas [125].

Todas estas herramientas son válidas para comprobar bases de reglas. Puesto que actualmente se están elaborando bases de conocimiento mixtas, se hace todavía, si cabe, mucho más necesario un módulo de validación y verificación para la base de conocimiento. En el caso de sistemas híbridos de reglas y marcos, se pueden presentar ciertas anomalías cuando se pone en marcha el mecanismo de la herencia de propiedades, por lo que se ha empleado una técnica que detecta inconsistencias en una base de conocimiento mixta [160].

Hasta ahora muchos autores coinciden en declarar diferencias entre las validaciones y verificaciones que conciernen al desarrollo de la base de conocimiento, y las relacionadas con la explotación de la misma, estableciendo distintas funciones a realizar en un proceso y en otro [205]. De forma equivalente, Preece establece, para bases de reglas, dos tipos de propiedades: propiedades estáticas, referentes a la base de conocimiento propiamente dicha, para las que también elabora una serie de controles de validación y verificación, y propiedades dinámicas, referentes al encadenamiento propiamente dicho [141]. De aquí deducimos que una base de conocimiento debe ser consistente, teniendo en cuenta dos aspectos: desde el punto de vista de la edición de cada unidad de conocimiento por separado y desde el punto de vista global, cuando se encadenan unas unidades con otras. Este

segundo aspecto, resulta más complejo de controlar, ya que engloba toda la base de conocimiento al completo. Así algunas propuestas consisten en elaborar previamente los caminos posibles de las reglas y a continuación comprobar la consistencia de los mismos, tal como ocurre en PATH HUNTER [71].

## 2.5. Propagación de la incertidumbre

Durante esta fase se lleva a cabo la combinación de los valores numéricos proporcionados por el usuario, junto con los que ya tenía el sistema, para llegar a una solución numérica. En el caso de que se desee asociar incertidumbre al sistema a desarrollar, deben pedirse los datos numéricos acordes con la teoría de propagación de la incertidumbre, que se vaya a emplear.

Se han realizado numerosos estudios acerca de las diferentes formas de manejar la incertidumbre [73]. Durante finales de la década de los 70, y a lo largo de los 80, aparecieron diferentes teorías para el manejo de la incertidumbre en S.E., destacando la teoría de la evidencia [158] y un sinfín de aproximaciones matemáticas a la misma [11, 175, 198, 54, 170]; la probabilidad subjetiva, implementada por primera vez en PROSPECTOR [55]; la teoría de los factores de certeza implementada en MYCIN [162] y algunas de sus aproximaciones [78]; la teoría de la posibilidad [201], la lógica difusa [202] y las redes bayesianas [133]. Todo este conjunto de teorías y sus aproximaciones han proporcionado soluciones cuantitativas para S.B.C. que manejan un sólo tipo de representación del conocimiento.

Durante la última década, con la creación de las bases de conocimiento híbridas, surge la necesidad de implementar más de una teoría de manejo de la incertidumbre en estas herramientas, con lo que esto permite elegir, en el momento de la propagación, la teoría que más se amolde al problema planteado. Walley establece una serie de criterios para clasificar los métodos de manejo de la incertidumbre en distintas categorías, según su precisión a la hora de propagar el conocimiento cuantitativo [178]. Así, el método de los factores de certeza se considera poco consistente aunque de fácil implementación, mientras que el método probabilístico se considera poco preciso pero consistente [178]. Herramientas como PULCINELLA [152] integran en su arquitectura tanto propagación de probabilidades, como posibilidades, valores booleanos y funciones de creencia [196].

La utilidad de sistemas con manejo mixto de la incertidumbre radica, no sólo en la simple solución cuantitativa a la que se puede llegar una vez propagados los valores, sino en la posibilidad de poder realizar un estudio comparativo de los resultados alcanzados por las diferentes teorías [152].

A continuación se describen las teorías de manejo de la incertidumbre que han tenido más repercusión durante las últimas décadas.

## 2.5.1. Probabilidad subjetiva

El método de la probabilidad subjetiva, implementado por primera vez en PROSPECTOR [55], pretende imitar el proceso de razonamiento de geólogos experimentados, para señalar lugares o regiones para la prospección de depósitos minerales del tipo representado por el modelo que el usuario en cuestión determine. Estos modelos agrupan una porción de conocimiento experto, relacionando depósitos minerales existentes y la información disponible acerca de ellos.

Por último describiremos los fundamentos matemáticos en que se basa esta teoría así como una breve descripción de su implementación informática.

### 2.5.1.1. Fundamentos generales

Ante la necesidad de codificar y clasificar los distintos modelos, se han realizado diferentes formalismos en PROSPECTOR:

- a) Redes inferenciales de asertos (afirmaciones).- Consisten en redes donde se establecen relaciones y conexiones entre las evidencias observadas y las hipótesis que derivan de las mismas. Puesto que no siempre son perfectamente distinguibles las evidencias de las hipótesis, se utilizará el término general “aserto” para hacer referencia a las primeras.
- b) Relaciones.- En PROSPECTOR se emplean tres tipos diferentes de relaciones para especificar cómo un cambio en la probabilidad de un aserto, afecta a la probabilidad de otros asertos. Se distinguen: Relaciones lógicas, relaciones plausibles y relaciones de contexto. A continuación pasamos a describir cada una de ellas:
  - b.1) Relaciones lógicas.- En este tipo de relaciones, la verdad o falsedad de una hipótesis está totalmente determinada por la verdad o falsedad de los asertos que la definen, puesto que tales relaciones están compuestas por las conectivas lógicas: AND, OR y NOT. Para tratar este tipo de relaciones, en el ámbito de la informática, se emplean las fórmulas de los conjuntos borrosos de Zadeh [200], de forma que, usando estas fórmulas, la probabilidad de una hipótesis cuyos asertos están unidos por AND, equivale al mínimo de los valores probabilísticos de los asertos observados; si están unidos por OR, el valor probabilístico de la hipótesis resultante, es el máximo de las probabilidades de los asertos que la componen. En algunas ocasiones, no existen todos los asertos que llevan a una hipótesis, afirmados o negados, sino que falta alguno, en este caso, se hace uso de las relaciones plausibles.

b.2. Relaciones plausibles. - El razonamiento plausible en PROSPECTOR se basa en la teoría bayesiana de la decisión en la que se introducen dos términos: **medida de suficiencia** y **medida de necesidad**.

La medida de suficiencia (MS), también denominada razón de verosimilitud positiva, es un término que relaciona dos elementos: la evidencia observada (E) y la hipótesis a deducir (H), en otras palabras, relaciona una medida previa sobre la hipótesis,  $\Phi(H)$ , denominada la razón de probabilidad previa o **radio probabilístico**, y una medida posterior sobre la hipótesis dada una observación previa:  $\Phi(H/E)$ , denominada la **razón de probabilidad posterior**. La fórmula viene a ser:

$$MS = \frac{\Phi(H/E)}{\Phi(H)}$$

Se pueden intercambiar las probabilidades y el radio probabilístico a través de la relación:

$$\Phi = \frac{P}{1-P} \quad \text{siendo P la probabilidad}$$

$$P = \frac{\Phi}{1-\Phi}$$

Otra forma de expresar la medida de suficiencia, viene dada por la fórmula:

$$MS = \frac{P(E/H)}{P(E/\bar{H})}$$

De aquí se extraen diferentes conclusiones según el valor de MS:

- Un valor alto de MS, significa que la evidencia apoya la hipótesis considerada.
- Un valor de MS que tienda a infinito, significa que la evidencia apoya totalmente la hipótesis considerada.
- Un valor menor que 1, significa que la evidencia apenas apoya la hipótesis considerada.

La medida de necesidad (MN), también denominada razón de verosimilitud negativa, es un término que relaciona dos elementos: la evidencia no observada y la hipótesis a deducir, en otras palabras, y como en el caso anterior, relaciona una medida previa sobre la hipótesis,  $\Phi(H)$ , denominada la razón de probabilidad previa, y una medida posterior sobre la hipótesis dado que no se dispone de una determinada observación previa,  $\Phi(H/E)$ ; la fórmula viene a ser:

$$MN = \frac{\Phi(H/E)}{\Phi(H)}$$

Otra forma de expresar la medida de necesidad, sería mediante la fórmula:

$$MN = \frac{P(\bar{E}/\bar{H})}{P(\bar{E}/H)}$$

de donde se pueden obtener una serie de reglas, según el valor de MN:

- Un valor de MN muy alto, significa que la evidencia observada, apenas apoya la hipótesis considerada.
- Un valor de MN próximo a cero, significa que la evidencia apoya totalmente la hipótesis considerada.
- Un valor de MN menor que la unidad, significa que la evidencia apoya la hipótesis considerada.

b.3) Relaciones de contexto.- Se emplean cuando las relaciones no se pueden usar de una forma arbitraria, sino que precisan un cierto orden secuencial, es decir, el que se dé un determinado aserto, depende de que se dé otro aserto anterior.

### 2.5.1.2. Implementación

El método PROSPECTOR está en línea con las técnicas bayesianas subjetivas propuestas en MYCIN e incluye variantes conceptuales significativas en cuanto a las medidas a proporcionar por los expertos y su forma de proceso. Por ejemplo como medida de implicación se utiliza la razón de probabilidad de una hipótesis, definida a partir de las probabilidades bayesianas y que viene a ser una medida análoga al factor de certeza empleado en MYCIN. Uno de los inconvenientes del método

PROSPECTOR, si lo comparamos con MYCIN, es que al utilizar una medida de certeza basada tan directamente en la probabilidad, supone que afirmar algo equivale a negar la fórmula complementaria, lo que en muchos casos del conocimiento real no es cierto [44].

Por otro lado, con frecuencia cuesta obtener valores exactos para probabilidad 'a priori'. En muchas áreas de lo único que se dispone es de estimaciones subjetivas, por lo que resulta difícil modificar un conjunto bayesiano de valores, debido a las dependencias entre ellos; siendo así, que las probabilidades de todas las hipótesis posibles que revelan la evidencia E deben sumar 1.0; si añadimos una nueva hipótesis, es fácil que tengamos que volver a calcular muchos valores de MS y MN. Cuando las fracciones de evidencia son dependientes, deben calcularse valores MS y MN para reflejar la presencia o ausencia de todas las combinaciones de estas fracciones de evidencia. El valor de probabilidad único asignado a una hipótesis no indica nada acerca de su precisión. El valor único combina la evidencia en favor o en contra de una hipótesis sin indicar la proporción que hay de cada una.

## 2.5.2. Factores de certeza

MYCIN es un S.E. de diagnóstico médico basado en reglas de producción y desarrollado por Shortliffe y sus colaboradores en la Universidad de Stanford [161]. MYCIN emplea, como método de tratamiento de la incertidumbre, la teoría de los factores de certeza con ciertas aproximaciones.

Para varios autores, MYCIN ofrece algunas lagunas a la hora de evaluar la incertidumbre, por lo cual ha habido varios intentos de dar interpretaciones probabilísticas a dicha teoría [83, 78].

MYCIN se considera un clásico en la Inteligencia Artificial. Su valor histórico es indudable, y se puede afirmar que ha sentado un precedente al ser el primer sistema experto que implementó en su momento un método de tratamiento de la incertidumbre, basándolo en un proceso de encadenamiento hacia atrás, que selecciona las reglas a aplicar [159].

Este método de manejo de la incertidumbre será recomendable aplicarlo en caso de S.E. basados en reglas de producción, orientados al diagnóstico o a la taxonomía, ya que presenta varias ventajas [162], tales como:

- Permitir el uso de conocimiento general, disponible en cualquier libro, para extraer conclusiones en casos particulares.
- Emplear conocimiento específico, en el supuesto de casos menos comunes.
- Ser fácilmente modificable.
- Detectar fácilmente las inconsistencias y contradicciones de la base de conocimiento.
- Proporcionar explicaciones durante la sesión de conocimiento.

El modelo de razonamiento inexacto, implementado en MYCIN, es una aproximación a la probabilidad condicional, y puede aplicarse a cualquier área de conocimiento, aunque según autores, no se han validado resultados de este método en otro tipo de aplicaciones [83].

El problema de diagnóstico puede resolverse con una gran cantidad de datos acerca de cada uno de los diagnósticos por separado, y acerca de cada uno de dichos diagnósticos, sabiendo que se poseen una serie de síntomas determinados. De esta forma, el teorema de Bayes puede aplicarse, considerando, como ejemplo, que 'Di' es el diagnóstico realizado, 'p(Di/E)' es la probabilidad condicional de que el paciente tenga una determinada enfermedad, conociendo una serie de síntomas agrupados por 'E':

$$P(Di / E) = \frac{P(Di) P(E / Di)}{\sum_{j=1}^n P(Dj) P(E / Dj)}$$

Pero esta solución es insuficiente cuando se trata de analizar síntoma a síntoma observado e ir incrementando numéricamente, la probabilidad del diagnóstico realizado, ya que no es lo mismo una conclusión extraída por la observación de un sólo síntoma, que la que se extrae por la observación de varios. Así, los S.B.C orientados a diagnóstico, que tienen en cuenta observaciones del mundo real, suelen utilizar una versión modificada del teorema de Bayes que es, en cierto modo apropiada, para ir incrementando probabilidades, según se observan síntomas. Así, siendo 'E1' el conjunto de todas las observaciones analizadas, 'S1' los nuevos datos, y 'E' el conjunto de observaciones, una vez que 'S1' se ha añadido a 'E1', tenemos:

$$P(Di / E) = \frac{P(S1 / Di \& E1) P(Di / E1)}{\sum_{j=1}^n P(S1 / Dj \& E1) P(Dj / E1)}$$

Pero este tipo de aplicaciones requiere una gran cantidad de datos. El encontrar un método de manejo de tratamiento de la incertidumbre adecuado para problemas relacionados con diagnóstico es difícil, puesto que las asociaciones entre un síntoma particular y su diagnóstico correspondiente, no son fijas sino que pueden cambiar con el tiempo a medida que se van añadiendo nuevos síntomas y nuevas observaciones en este campo.

El teorema de Bayes sería apropiado en caso de disponer de una gran cantidad de datos. La cuantificación formal de las probabilidades asociadas a la toma de decisiones en diagnóstico, es poco veraz, puesto que la diagnosis no es un proceso determinístico; es necesario por ello desarrollar un modelo que emplee una aproximación del análisis bayesiano y probabilístico y es esta idea, la que se

ha implementado en el corazón de MYCIN [161]. Los desarrolladores de MYCIN proponen trabajar con reglas de decisión como si fueran cajas cerradas de conocimiento y con un método de cuantificación que permita la acumulación de la evidencia observada. Por tanto la teoría de la certidumbre surge como alternativa a la teoría de la probabilidad con el fin de solucionar algunos de los problemas reflejados en ésta última [62].

El método más sencillo para representar la incertidumbre, consiste en emplear un valor único que refleje la veracidad de una determinada conclusión. En la teoría de la probabilidad, este valor representa la probabilidad a priori; después el valor es actualizado, utilizando reglas bayesianas. Este método está implementado en el S.E. PROSPECTOR, pero el emplear un valor único no indica nada acerca de la magnitud por separado de la evidencia a favor o en contra de dicha conclusión [62].

Un enfoque alternativo es mantener dos valores por separado para cada conclusión: medida de creencia e increencia de la hipótesis en estudio, y combinar ambas en un sólo valor, que se conoce con el nombre de factor de certeza. Dichos factores de certeza se asocian a reglas y proporcionan una medida de la fiabilidad de las mismas, teniendo unos valores comprendidos entre -1 y 1. Si se aplican dos reglas contradictorias, de tal forma que la certidumbre de una sea igual a la de la otra, entonces sus efectos se destruyen [62].

### 2.5.2.1. Fundamentos generales

Los factores de certeza constituyen una medida distinta de la probabilidad, ya que los grados de certeza que afirman y niegan una hipótesis, son opuestos y no suman uno, como ocurre cuando se habla de probabilidades.

La medida de creencia es el incremento que la evidencia 'e' proporciona a la creencia que se tiene acerca de una hipótesis concreta 'h', [161].

$$MB[h, e] = \begin{cases} 1 & \text{si } P(h) = 1 \\ \frac{\max[P(h/e), P(h)] - P(h)}{\max[1, 0] - P(h)} & \text{otro caso} \end{cases}$$

La medida de increencia es la disminución que la evidencia 'e' proporciona a la creencia que se tiene acerca de una hipótesis concreta 'h', [161].

$$MD[h, e] = \begin{cases} 1 & \text{si } P(h) = 0 \\ \frac{\min[P(h/e), P(h)] - P(h)}{\min[1, 0] - P(h)} & \text{otro caso} \end{cases}$$

El factor de certeza es un número resultante de la combinación de las dos medidas anteriores que da una idea de la veracidad o ignorancia acerca de una determinada hipótesis, dada una evidencia sobre ella [161]:

$$CF[h,e] = MB[h,e] - MD[h,e]$$

Los factores de certeza para considerarse como tales, han de cumplir tres axiomas fundamentales:

- Ser números reales.
- Tomar cualquier valor entre -1 y +1, es decir, se puede asegurar una continuidad entre estos dos valores extremos.
- Cumplir el **axioma de la modularidad**, [78], según la cual, toda actualización de una creencia depende de tres elementos: la hipótesis que se está actualizando, la evidencia que provoca tal actualización, y otra evidencia, conocida en el momento de la actualización, pero que cuyo efecto se considera nulo, es decir, se asume:

$$CF(H,E,e) = CF(H,E,\emptyset) \equiv CF(H,E),$$

para cualquier evidencia 'e' que se podría conocer al tiempo que E actualiza H.

El axioma de la modularidad asegura la construcción y el mantenimiento de la base de conocimiento, es decir, las reglas se pueden añadir y borrar de la base de conocimiento sin la necesidad de considerar las interacciones con el resto de las reglas [83].

### 2.5.2.2. Implementación

A partir del desarrollo de la teoría de los factores de certeza y su implementación en MYCIN, han sido numerosos los sistemas que han implementado este mecanismo de manejo de la incertidumbre. Dan Qiu establece una implementación de esta teoría añadiendo una tercera medida: la medida de necesidad, que cuantifica el grado de necesidad que se precisa para que se establezca la hipótesis sobre la que se está actuando [142]. La construcción de la herramienta MOLE, permite elaborar los factores de certeza de una forma automática, sin necesidad de extraerlos directamente del experto [58].

### 2.5.3. Redes bayesianas

Una red bayesiana es un grafo dirigido acíclico donde los nodos representan variables aleatorias y los arcos representan las relaciones de dependencia condicional entre dichas variables. Fueron introducidas por Pearl [133] con el fin de proporcionar una forma de combinar una semántica probabilística y declarativa [101] a un conjunto de variables que presentan dependencias entre ellas. Las redes bayesianas, en la última década, han pasado a ser una forma de representación común para codificar todo cuanto se refiere a conocimiento experto incierto en S.B.C. [79]. Las redes bayesianas ofrecen información acerca de las relaciones causales entre las variables, proporcionan una forma de manejar el conocimiento incierto y combinan conocimiento previo y datos.

Las redes bayesianas combinan la teoría de la probabilidad con la representación gráfica del dominio del caso de estudio. La teoría de la probabilidad ofrece un marco apropiado para tratar la incertidumbre y la representación gráfica facilita de una manera rápida, el detectar dependencias e independencias existentes en el dominio del conocimiento, haciendo más sencilla la adquisición del conocimiento y su representación.

Es necesario diferenciar entre probabilidad clásica, como probabilidad universal, que para asignarla es necesario repetir el experimento un número ilimitado de veces, y la probabilidad bayesiana, que es totalmente subjetiva y pertenece a la persona que la asigna [79].

Las redes bayesianas son una forma de representación adecuada cuando se quiere tratar un gran número de variables. Una de las grandes ventajas que presentan es que cualquier cambio en la configuración de la red es admitido sin problemas por la red bayesiana [135].

La semántica causal de las redes bayesianas, es en parte responsable del éxito de las mismas, de ahí que sea necesario disponer de una semántica completa, independiente del lenguaje de propagación. Dicha semántica servirá como lenguaje de representación de la base de conocimiento, con el fin de conocer el significado preciso de cada proposición representada en la red y con el fin de que el algoritmo de generación de la red sea correcto [74].

De esta forma a medida que han ido evolucionando las técnicas de programación y con la aparición del paradigma de la orientación a objetos, ha comenzado a desarrollarse una nueva semántica en las redes bayesianas: las redes bayesianas orientadas a objeto, según las cuales, las relaciones probabilísticas entre las variables son en realidad relaciones probabilísticas entre los atributos de un mismo objeto [101]. En muchas ocasiones resulta difícil extraer valores numéricos a los expertos para completar la red, en este caso se puede llevar a cabo una estimación de parámetros con el fin de completar los valores no conocidos [12].

### 2.5.3.1. Fundamentos generales

La construcción de las redes bayesianas, conllevan las siguientes fases:

- a) Fase 1<sup>a</sup>.- Identificar las variables relevantes del área a tratar, tanto observaciones, como metas.
- b) Fase 2<sup>a</sup>.- Integrar dichas variables y metas, estableciendo entre ellas relaciones causales.
- c) Fase 3<sup>a</sup>.- Asignar distribuciones locales de probabilidad.

El resultado final consiste en una red bayesiana constituida por un conjunto de variables:  $X=\{x_1, x_2, \dots, x_n\}$ , reunidas en una estructura en red que codifica un conjunto de afirmaciones condicionales e independientes entre sí en X, y un conjunto P de distribuciones de probabilidad asociadas con cada variable. Ambas componentes definen una distribución de probabilidad conjunta.

La estructura de una red bayesiana está formada por nodos y arcos. Los nodos representan el dominio de una variable con valores mutuamente exclusivos y exhaustivos. Se relacionan con cada variable mediante una correspondencia uno a uno.  $x_i$  identifica una variable concreta en X y su nodo en la estructura, y  $P_{ai}$  denota los padres del nodo  $x_i$ , así como las variables a que hace alusión. La distribución de probabilidad conjunta viene dada por:

$$p(x) = \prod_{i=1}^n p(x_i / p_{a_i})$$

Las redes bayesianas son una forma de representar conocimiento cuantitativo y cualitativo mediante una sola estructura. Los grafos dirigidos acíclicos sirven para representar el conocimiento cualitativo, mientras que las probabilidades condicionadas representan el conocimiento cuantitativo siendo ésta precisamente la parte propia del experto, donde se pone de manifiesto la experiencia del mismo a la hora de asignar dichos valores a cada uno de los conocimientos introducidos. El modelo general, por tanto, consta de un conjunto de variables, constituyendo cada una un nodo del modelo causal, disponiendo éste de un conjunto de nodos ‘padre’ con los que guarda una relación dirigida. Asimismo, existe una tabla de probabilidades condicionadas para cada uno de los nodos ‘padre’ referidos a los nodos ‘hijo’ que posee. Dada esta estructura se puede dar la actualización secuencial de las probabilidades condicionadas sobre estructuras dirigidas [165].

### 2.5.3.2. Propagación

Inicialmente los primeros algoritmos de propagación de las redes bayesianas se basaban en el paso de mensajes y se limitaban sólo a árboles y redes simplemente conectadas. En cuanto se comenzaron a desarrollar redes múltiplemente conectadas, aparecieron numerosos algoritmos de propagación, según se hacía diferente hincapié en unos u otros aspectos de la red.

Así, los métodos de propagación en redes bayesianas, responden a tres tipos:

- **Métodos de propagación exacta.**- Los cálculos de las probabilidades cuentan sólo con los errores producidos por el propio redondeo ocasionado por las limitaciones del ordenador
- **Métodos de propagación aproximada.**- Disponen de algoritmos de cálculo para obtener los valores de las probabilidades.
- **Métodos de propagación simbólica.**- Se basan en la obtención de las probabilidades en función de parámetros.

Entre los métodos de propagación exacta, destacan los apropiados para poliárboles, simplemente conexos, denominados métodos de propagación en poliárboles [98, 133]. Por otro lado destacan los apropiados para poliárboles múltiplemente conexos, que se agrupan en dos tipos, los algoritmos de condicionamiento [136, 157] y los algoritmos de agrupamiento [107, 166, 52].

### 2.5.3.3. Implementación

Durante la última década se ha producido un desarrollo de S.B.C. que implementan redes bayesianas, así como de herramientas de propósito general preparadas para implementar cualquier tipo de red bayesiana. Entre las primeras destacan:

- **PAINULIM** [192].- Se trata de un S.E. para el diagnóstico de enfermedades neuromusculares. Se ha empleado una red bayesiana para su desarrollo que contiene 83 variables representando 14 enfermedades y 69 síntomas, cada uno de los cuales con tres posibles valores. Debido a que la red que se crea está múltiplemente conectada, con 271 arcos y 6795 valores de probabilidad, la técnica desarrollada para propagar la incertidumbre en esta red, ha sido la red bayesiana múltiplemente seccionada (MSBN), consistente en extraer a partir de la red bayesiana original, las subredes existentes y transformarlas en árboles de unión [193, 194].
- **DIAVAL** [53].- Representa un S.E. para Ecocardiografía, que consta de 324 nodos, entre los nodos de la red, el grupo más numeroso consta de 325 datos, de los cuales, 52 son parámetros. Estos nodos se unen entre sí por 335 enlaces, que responden a diferentes tipos.

La técnica desarrollada para propagar la incertidumbre en esta red, ha sido una extensión del algoritmo de propagación exacta en políárboles de Kim [98], introducida por el autor, consistente en un algoritmo de condicionamiento local y el uso de la puerta OR graduada.

Entre las herramientas de propósito general, destacan:

- BANTER [75].- Ofrece una herramienta que sirve para visualizar la información asociada a una red bayesiana, de una manera fácil para un usuario final. La única condición es que los nodos de la red bayesiana introducida puedan clasificarse en hipótesis, observaciones y diagnósticos, no necesitando el usuario conocer nada acerca de redes bayesianas, tan sólo debe tener conocimientos del área a tratar. El sistema proporciona también diferentes formas de elaborar preguntas a la base de conocimiento, así como petición de explicaciones en un momento dado.
- HUGIN [9].- Se trata de una herramienta que ayuda a construir redes bayesianas aplicables a sistemas expertos. HUGIN permite la construcción de modelos que pueden ser luego usados en otras aplicaciones, se dice que HUGIN es la herramienta ideal para educación, por su manejo intuitivo. Posteriormente a su desarrollo ha sido ampliado con diferentes algoritmos de manejo de la incertidumbre. Así HUGIN proporciona un entorno de propagación de la incertidumbre para el caso de tratar con redes bayesianas de creencia [107], o bien casos de independencia condicional [134], diagramas de influencia [93] y redes de creencia que manejan variables condicionales gausianas [108].

# **Capítulo 3**

---

## **Tecnología de Objetos**

### **3.1. Orígenes de la orientación a objetos**

El término de 'programación orientada a objetos' deriva del concepto de objeto existente en el lenguaje de programación 'Simula 67'. El antecesor de este lenguaje fue Simula I, que surge como lenguaje de simulación. Simula I se diseñó entre 1962-1964, aunque no estuvo disponible hasta 1965, en que se consideró ya no sólo como lenguaje de simulación, sino lenguaje de propósito general. Simula 67 alcanzó gran éxito al ser capaz de proporcionar un tipo de dato: el cluster, que agrupaba datos y procedimientos. La forma de acceder a los datos era mediante los procedimientos creados expresamente para dicho cluster, de lo contrario, no se permitía el acceso. Esta forma de programar, adelantó el nacimiento tanto de uno de los principios de la orientación a objetos: el encapsulamiento como del concepto de clase.

Según Mathews [115] el enfoque orientado a objetos (O.O.) arranca de los llamados sistemas holónicos por la relativa autonomía de los integrantes celulares u holones, por su cierta dependencia del resto del sistema y por su recursividad en cierto grado. El término de ‘holón’ fue introducido por primera vez por Koestler [100] haciendo alusión a algo que es incompleto en el sentido de que forma parte de un todo, pero que por otro parte engloba en sí mismo todo lo necesario para ser autónomo. Este concepto sirvió de gran utilidad sobre todo en el contexto del funcionamiento de los sistemas biológicos. Estos holones vendrían a ser las unidades de actuación de la programación O.O: los objetos, a los que se puede acceder desde cualquier parte del programa mediante el envío de un mensaje; por tanto a los objetos se les empieza a dotar de toda la funcionalidad necesaria para que, de la interacción de unos con otros, originen como resultado la elaboración de un sistema informático.

El paradigma de la orientación a objetos surge como alternativa a la visión estructurada que venía protagonizando el desarrollo de los sistemas informáticos desde tiempo atrás. La orientación a objetos plantea una nueva forma de enfocar un problema. Su objetivo es modelar la realidad empleando conceptos del mundo real , en contraste con paradigmas anteriores que ponían especial acento en la descomposición funcional y no tanto en la conceptual. De esta forma, el paradigma de la orientación a objetos, centrado en la identificación de los objetos, genéricamente clases, aporta los siguientes elementos:

- Clase.- Representa a un conjunto de datos y operaciones. Una clase constituye realmente una plantilla o un tipo de datos capaz de dar lugar a la creación de objetos.
- Superclase.- Clase que engloba un conjunto de clases al compartir con ellas algunos datos y operaciones; una superclase se encuentra en un nivel jerárquico por encima de otras clases.
- Subclase.- Clase perteneciente a una clase superior. Una subclase se encuentra en un nivel jerárquico por debajo de otras clases que comparten con aquella algunos datos y operaciones.
- Atributo.- Propiedad o dato miembro de las clases.
- Objeto.- Conjunto de atributos de una clase que toman un valor concreto para cada uno de ellos.
- Método.- Operación o función miembro que hace que los objetos se relacionen entre sí.
- Agregación.- Relación de composición entre unas clases y otras.

El enfoque orientado a objetos constituye una nueva forma de concebir cualquier tipo de sistema; ofrece una visión más modular y favorece la reutilización, al formalizar un sistema basado en la interacción entre los objetos; se basa en los siguientes principios esenciales:

- Abstracción - Consiste en centrarse en las propiedades esenciales de un objeto y olvidar momentáneamente las propiedades accidentales.
- Encapsulamiento.- Trata de separar los datos propios de los objetos de las formas de acceso a los mismos.

- Herencia.- Permite compartir datos y funciones miembro entre clases, gracias a la existencia de las relaciones jerárquicas.
- Polimorfismo.- Determina los comportamientos diversos que puede adquirir una misma función miembro.

La orientación a objetos proporciona unos planteamientos más cercanos al pensamiento humano, ya que el hombre basa su razonamiento en la clasificación de todo cuanto le rodea, y en las relaciones; el enfoque orientado a objetos, mediante los mecanismos de abstracción, clasifica, y mediante la herencia, establece las interacciones de unas clases con otras. Este paradigma ha sido revolucionario en tanto en cuanto ha proporcionado un enfoque totalmente distinto de los sistemas informáticos, aunque ya Nygard en 1986 aventura la aparición de otros paradigmas tales como la programación orientada a las funciones o la programación orientada a las restricciones [130].

## 3.2. Evolución de lenguajes

La aparición de lenguajes orientados a objetos es consecuencia de una evolución debida a dos factores:

- Cambios en los paradigmas de la programación
- Cambios en la arquitectura del hardware

Los objetivos que quieren lograrse cuando se programa una determinada aplicación informática han ido cambiando con el tiempo. Desde el paradigma original de programación, dirigido a los procedimientos, pasando por el paradigma centrado en los módulos, o bien, más adelante centrado en los tipos de datos, se han ido concibiendo de manera paralela diferentes lenguajes de programación, o evoluciones de los mismos, con el fin de que se adapten al paradigma reinante en ese momento. Finalmente el paradigma de programación existente en la actualidad es el que se dirige hacia las clases también llamado orientado a objetos.

Inicialmente lenguajes tradicionales como FORTRAN (FORmula TRANslation) desarrollado por Backus en 1954 o ALGOL (ALGOrithmic Language), desarrollado en Europa en 1958, casi simultáneamente al anterior, han ido contribuyendo a la aparición de otros lenguajes más evolucionados y adaptados a las nuevas tecnologías.

De esta forma, ALGOL-58, más tarde ALGOL-60, interviene en el desarrollo de otros lenguajes como SIMULA, CPL y ALGOL-68. CPL, de cuyas iniciales no está muy claro su significado: 'Cambridge o Combined Programming Language', ha destacado por contribuir al nacimiento de

BCPL (Basic CPL), concebido por Martin Richard en 1967, que a su vez, dio lugar al lenguaje B, escrito por Ken Thompson en 1970 y cuyo objetivo primordial fue reescribir UNIX, que inicialmente estaba escrito en ensamblador. La modificación del lenguaje B por Dennis Ritchie dio lugar a la aparición de C [96], que entre otras cosas sirvió para reescribir UNIX. CPL también destaca por ser precursor de PASCAL, escrito por Niklaus Wirth [188] y cuyo objetivo era fundamentalmente pedagógico: enseñar a los alumnos la tarea de la programación. PASCAL ha dado lugar a dos desarrollos relacionados ya con el paradigma de la orientación a objetos: por un lado ADA, desarrollado en 1979 por el departamento de defensa americano y estandarizado en 1983 y por otro Turbo Pascal O.O [106]. En la figura 3.1. se detallan los orígenes de lenguajes orientados a objetos como TPascal OO, ADA o C++.

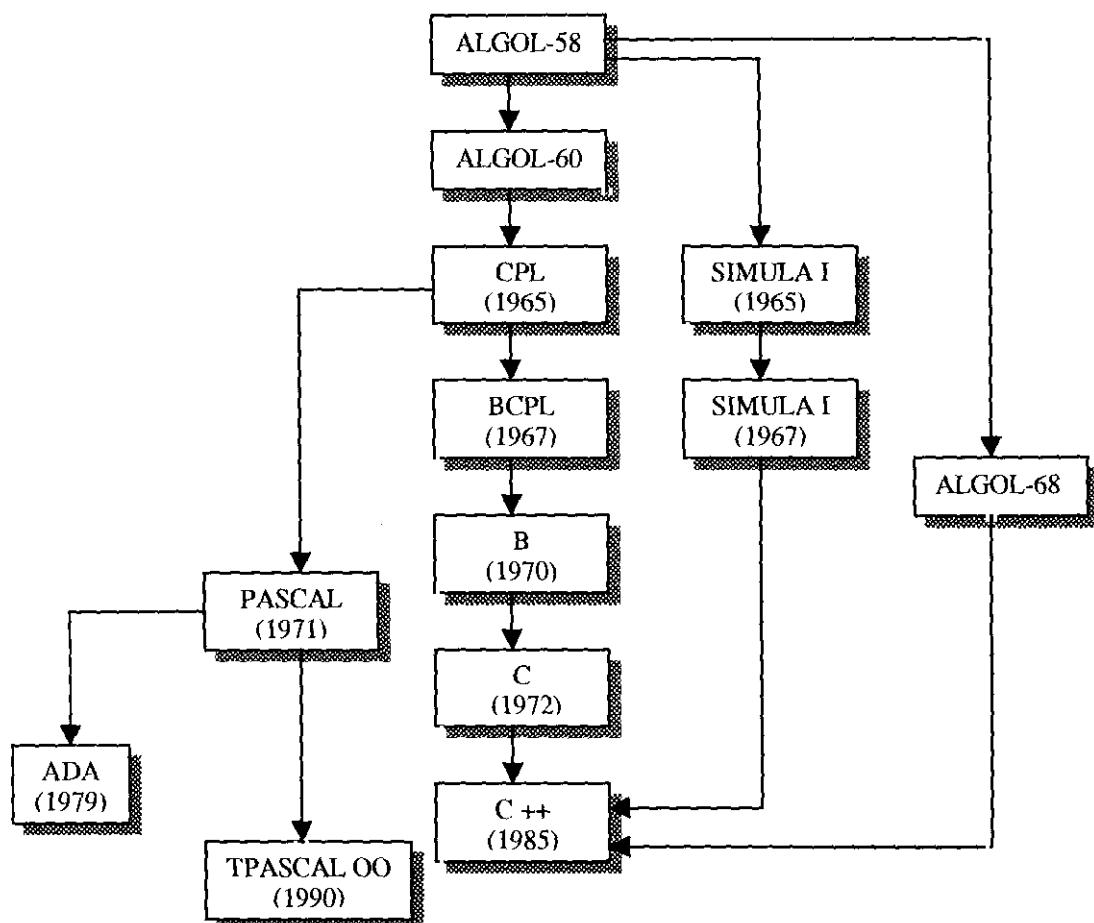


Figura 3.1. Evolución de lenguajes de programación

Más adelante, con la proliferación de ordenadores personales, poseedores de elementos gráficos que hacen más sencillo el manejo de las herramientas informáticas, la programación tradicional aportada por lenguajes C debe sustituirse por una programación basada en objetos y mensajes. Así

Allan Kay, Adele Goldberg, Larry Tesler, entre otros, se unen para crear otro lenguaje que implemente los principios de la programación orientada a objetos: SMALLTALK-80. SMALLTALK inspira una familia de lenguajes orientados a objetos tales como CLOS (Common LISP Object System), extensión de LISP [16] o L&O (Logic & Objects) extensión de PROLOG. CLOS es un lenguaje que permite la creación dinámica de objetos y que en contraste con LISP permite la herencia múltiple.

Según van avanzando las nuevas tecnologías, la demanda de desarrollo de aplicaciones visuales es tan alta, que es difícil cubrirla con los programadores existentes no habituados todavía a la programación orientada a objetos. Surge una solución que es la reutilización, aunque no se sabe cómo llevar a cabo. Existen demasiadas personas que programan en C y FORTRAN y resultaría caro no aprovecharlas. De esta forma en 1980 se añaden ‘clases’ al lenguaje C, convirtiéndolo en ‘C con clases’, y posteriormente en 1985, Stroustrup se aventura a crear un lenguaje híbrido, que tenga la posibilidad de soportar una programación orientada a objetos, sin perder la capacidad estructurada de C, llamándolo C++ [167]. Su nombre se debe a Rick Mascitti y hace alusión al operador de incremento ‘++’ existente en el lenguaje.

### 3.3. Evolución de metodologías

Tradicionalmente, el desarrollo de las aplicaciones informáticas se ha venido realizando mediante el empleo de lenguajes de tipo estructurado, tales como Cobol o C [96]. Por otro lado las metodologías de análisis y diseño que se aplicaban para este tipo de lenguajes tenían como consecuencia un marcado carácter estructurado y funcional. Diferentes técnicas de análisis estructurado [199, 51, 65, 90, 179] eran también las que se empleaban para la elaboración de una documentación previa al desarrollo de la aplicación. Su unidad básica era el proceso y en ellas se trataba de vislumbrar, de manera secuencial, las relaciones funcionales del sistema.

Posteriormente, el paradigma de la orientación a objetos desencadena el desarrollo de metodologías O.O. tales como la de ‘responsabilidad-colaboración’ –RDD– [187], ‘Análisis Orientado a Objetos’ [33,34], ‘Diseño Orientado a Objetos’ [17], ‘Técnica de Modelado de Objetos’ [149], ‘Ingeniería de Software Orientado a Objetos’ –OOSE– [91], cuyos objetivos son facilitar el desarrollo de aplicaciones bajo un enfoque orientado a objetos. Si comparamos la aparición de las metodologías de desarrollo orientado a objetos con la de los primeros lenguajes orientados a objetos, parece existir un cierto desfase temporal entre ambos, por lo que durante un tiempo, es inevitable que convivan lenguajes O.O. con metodologías de análisis estructurado.

De aquí surge la pregunta:

**¿Es posible integrar la orientación a objetos con un análisis y diseño estructurado? [180]**

Una integración de ambos consiste en realizar lo que concierne al módulo de captura de requisitos, según un análisis estructurado, y hacer el resto del análisis según un enfoque orientado a objetos [163]. La respuesta de George [67] al interrogante planteado por Ward consiste en proporcionar la correspondencia entre elementos de un análisis estructurado con elementos propios de la orientación a objetos. Una posibilidad es emplear los diagramas entidad-relación para extraer las clases e incluso las restricciones. Según otros autores tal integración sí es posible y hasta hace madurar el enfoque orientado a objetos; esto es lo que Hardgrave ha denominado con el término ‘evolución’. Para otros, no tiene sentido integrar un enfoque nuevo, novedoso, de análisis e implementación, con un enfoque tradicional; esto es lo que Hardgrave denomina ‘revolución’ [76]. Por tanto, parece que la convivencia de un enfoque de análisis estructurado con una programación informática orientada a objetos presenta evidentes contradicciones.

Curiosamente, a medida que nos adentramos en la década de los 90, las metodologías orientadas a objeto, van tomando cada vez más protagonismo, por lo que algunos autores han planteado estudios comparativos entre las más relevantes figurando Fichman [60] y Iivari [86] y otros desarrollan metodologías propias basadas en lo mejor de las ya existentes, como es el caso de FUSION [37]. Frecuentemente se plantea una metodología orientada a objeto en su totalidad, proponiendo la utilización de los casos de uso [92] para el módulo de captura de requisitos [150].

Xiaodong propone una metodología con dos componentes espaciales, una vertical para las necesidades y otra horizontal para el análisis, diseño, programación informática y pruebas, aunque claramente separa los requisitos del resto, no establece pautas claras para el diseño del prototipo preliminar [195]. Para llevar a cabo la captura de necesidades es más apropiado el enfoque estructurado, puesto que las necesidades de un sistema se expresan siguiendo una descomposición funcional más que conceptual. Dada la laguna existente en el mundo de los objetos, en cuanto a la captura de requisitos se refiere, Bailin formaliza un método de captura de especificaciones de un sistema, según un patrón de orientación a objetos [10]. Sin embargo, algunos autores declaran las carencias que supone analizar cualquier tipo de problema teniendo en cuenta tan sólo dicho enfoque y proponen como metodología de diseño una combinación entre la orientación a objetos y la lógica formal [114].

Paralelamente al desarrollo de las metodologías de análisis y diseño, surgen herramientas visuales de desarrollo, como Visual C++ de Microsoft, IVPE [84] que soportan lenguajes O.O., tales como C++ y que permiten la elaboración de aplicaciones con una apariencia gráfica basada en la programación conducida por eventos [137]. De esta forma, surge la necesidad de integrar en las metodologías de análisis, la manera de formalizar el diseño visual de cualquier aplicación a

desarrollar, así surge VMT [59] que proporciona una técnica de modelado visual basada en las metodologías existentes en el momento, añadiendo la componente gráfica al diseño para poder construir aplicaciones que dispongan de una interfaz de usuario; OOMGRIN [112], que combina casos de uso y modelos multiagente para desarrollar aplicaciones visuales; USE-IT [3] a partir de ciertas especificaciones mediante un sistema basado en el conocimiento, proporciona decisiones de diseño gráfico en la programación de la aplicación informática.

SOFL [111] es una metodología que integra métodos estructurados, métodos formales y metodologías orientadas a objeto. En ella las necesidades se concretan a lo largo del desarrollo del sistema, dividiendo la especificación de los mismos en dos tipos: requisitos primarios funcionales, especificados en las primeras fases del desarrollo y requisitos secundarios, especificados durante las fases de diseño y programación. Jaaksi [88] establece la captura de requisitos basada en un enfoque orientado a objetos para realizar el diseño de la interfaz gráfica de usuario (I.G.U.).

Otro tipo de desarrollos realizados con el objetivo de integrarse en las nuevas tendencias de la orientación a objetos, ha sido por ejemplo JASMINE [87]. JASMINE consiste en una base de datos orientada a objetos que surge como alternativa a las bases de datos relacionales para poder llevar a cabo el diseño y programación de objetos complejos. MEDEA [139] presenta una metodología para el análisis y diseño de bases de datos orientadas a objetos. SHERLOCK [113] chequea la consistencia de la I.G.U. mediante la evaluación de las propiedades de texto y visuales de la interfaz de usuario. FUZE [61] es otra herramienta que surge como consecuencia de la integración de FUSION con el lenguaje Z, subsanando de esta forma la ausencia de un lenguaje formal para el enfoque orientado a objetos de FUSION con la sintaxis y semántica aportada en el lenguaje Z.

La elaboración del análisis y diseño para una aplicación requiere de manera genérica, el conocimiento del lenguaje en que se va a programar, teniendo en cuenta que todas estas metodologías proporcionan el patrón general de elementos de la I.G.U. Wolber, sin embargo, propone un acercamiento al lenguaje de programación desde las fases de análisis, aportando una metodología para la descomposición funcional dentro del mundo de los objetos. Wolber, a partir de Visual C++ y su librería de clases, MFC, establece pautas para extraer jerarquías de eventos que van desde la I.G.U. hasta el interior de la aplicación [189]. En los diagramas de mensaje que emplea, utiliza enteramente la terminología de la herramienta, haciendo más complicada la elaboración y comprensión del análisis a un profano del lenguaje.

Por otra parte, cada vez son más numerosos los autores que se decantan por la técnica de modelado de objetos (OMT) [149] haciendo ciertas extensiones a la misma. Así, OST integra el modelo de objetos y el modelo funcional de OMT mediante una semántica formal [176], pero en cambio, emplea una diagramación poco intuitiva a simple vista. Como afirma Cockburn [36], una metodología es más efectiva cuanto más clara y más sencilla es de interpretar. Para Hasselbring, [77] OMT presenta carencias a la hora de cubrir los requisitos, así que diseña una herramienta --PROSET

LINDA-- para ayudar a analizar las necesidades, pero no plantea una forma de realizar diseños de la I.G.U., fácilmente deducibles del modelo de objetos de OMT

En busca de un estándar en la notación a seguir en la programación orientada a objetos, en 1997 surge el lenguaje unificado de modelado (UML) [18] como intento de aunar esfuerzos entre los precursores de diferentes metodologías O.O. para conseguir llegar a un estándar en cuanto a notación y semántica, de esta forma se consigue dar una cierta madurez y estabilidad a la orientación a objetos. Aunque una notación no es suficiente para la construcción de software [81], con UML se ha conseguido estandarizar todo el ciclo de vida de software, de forma que las herramientas de diseño existentes en la actualidad se ven obligadas a ofrecer UML entre sus opciones para llevar a cabo el proceso de ingeniería de software.

La metodología unificada sirve también, como base de otras metodologías que comienzan a surgir, así, Jaaksi [89] considera que todas las existentes en la actualidad son demasiado complicadas y familiarizarse con ellas requiere mucho tiempo, proponiendo la metodología simplificada, que emplea la notación de UML y que cubre todo el ciclo de vida de software en cinco fases. A partir de la tercera fase, o fase de diseño, la metodología parece dependiente de la herramienta visual a emplear en la programación, realizándose con ella todo el prototipo preliminar. A pesar de su sencillez, esta metodología adolece de una conexión exhaustiva de la I.G.U. con las entidades que surgen en las primeras fases del análisis. Para otros autores, una de las carencias en las metodologías existentes consiste en cómo manejar grandes volúmenes de clases cuando una aplicación tiene un cierto grado de complejidad. Así Zendler propone la realización de una taxonomía de objetos, según la funcionalidad que desempeñen en la aplicación, quedándose en un diseño de clases pero sin descender a nivel del comportamiento de dichas clases entre sí [203].

Basándonos en estas premisas podemos concluir que no es fácil hacer que convivan lenguajes y metodologías que se basen en diferentes paradigmas o estilos de programación, esto es: 'orientado al proceso' y 'orientado al objeto'. Y que al igual que en el apartado anterior se exponía cómo los lenguajes han ido evolucionando con el fin de adaptarse tanto a las nuevas tecnologías como a las nuevas necesidades, asimismo las metodologías de desarrollo no se quedan atrás buscando nuevas aproximaciones que incluya total o parcialmente el paradigma que se esté empleando en ese momento en el mundo del desarrollo de software.

### 3.4. Metodologías orientadas a objetos

A partir del desarrollo del paradigma de orientación a objetos, son muchas las metodologías que comienzan a aparecer con el fin de facilitar la tarea del análisis y diseño de una aplicación que se va a desarrollar bajo el enfoque de la orientación a objetos. En este apartado explicaremos algunas de

las metodologías y notaciones que han tenido un mayor éxito y que son más comúnmente utilizadas por los desarrolladores.

Según Henderson-Sellers [81] una metodología está constituida por un proceso de modelado, una semántica, unas técnicas que lo ponen en práctica para un caso concreto y un lenguaje que la representa haciendo uso de una determinada notación. De esta forma a continuación expondremos un conjunto de metodologías orientadas a objeto que consideramos las más relevantes actualmente. El último subapartado lo hemos dedicado al estándar UML, no considerado como metodología, sino como lenguaje, al ser compendio de varias metodologías y perseguir como principal objetivo la estandarización de la notación a emplear a la hora de hacer ingeniería de software.

Los criterios que se consideran los más importantes para seleccionar una u otra metodología varían, por ejemplo una metodología debe proporcionar un alto grado de flexibilidad, rigor y creatividad según Henderson-Sellers [80]. Amescúa propone cuatro criterios que, según él, una metodología debe cumplir para considerarse óptima. Estos son: versatilidad, funcionalidad, productividad y efectividad [8].

Toda metodología se apoya en un conjunto de diagramas que responden a una notación concreta. Las distintas metodologías O.O. que existen actualmente difieren entre sí en los aspectos que subraya cada una [50, 186], como veremos oportunamente en una breve descripción de las técnicas y metodologías que hemos considerado más interesantes por su relevancia en sí y por haber contribuido de alguna forma al desarrollo del estándar UML.

### 3.4.1. Metodología de Coad-Yourdon

En este apartado se expone una breve descripción de cuatro técnicas orientadas a objetos pioneras en el desarrollo de metodologías. En primer lugar, veremos el ‘Análisis orientado a objetos’ desarrollado por Coad-Yourdon [33,34] y que se conoce como O.O.A.(*Object Oriented Analysis*). El objetivo fundamental de O.O.A. consiste en la extracción del modelo de análisis para descubrir la funcionalidad del sistema; este método tiene las siguientes fases de desarrollo:

- a) Fase 1<sup>a</sup> - Consiste en la extracción de las clases y los objetos del sistema.
- b) Fase 2<sup>a</sup> - Durante esta fase se identifican las relaciones de generalización/especialización del sistema y las relaciones de composición.
- c) Fase 3<sup>a</sup> - Se trata de definir áreas, es decir, agrupar objetos según distintos criterios en una entidad superior denominada área.
- d) Fase 4<sup>a</sup> - En ella se identifican los atributos, que definen cada objeto.

- e) Fase 5<sup>a</sup>.- Consiste en la identificación de los servicios que definen las operaciones de las clases.

Una vez llevadas a cabo estas fases, se dispone de un modelo de análisis listo para llevar a cabo la programación informática del sistema.

### 3.4.2. Metodología de Booch

La metodología denominada ‘Diseño orientado a objetos’, también conocida por O.O.D. (*Object-Oriented Design*), desarrollada por Booch [17], presenta un gran parecido con la anterior, estando formada por una serie de fases, que se pueden resumir en las siguientes:

- a) Fase 1<sup>a</sup>.- Durante esta fase se identifican las clases y objetos a un nivel concreto de abstracción.
- b) Fase 2<sup>a</sup>.- Consiste en la identificación de las semánticas de dichas clases y objetos.
- c) Fase 3<sup>a</sup>.- En esta fase se identifican las relaciones entre clases y objetos.
- d) Fase 4<sup>a</sup>.- En ella se programan las clases y los objetos anteriormente extraídos.

Esta técnica ofrece un gran número de conceptos y diagramas para describir un sistema, lo que la convierte en una de las metodologías más completas. Algunos de estos diagramas son por ejemplo el diagrama de clases, el de objetos, el de tiempos, el de transición o el de estados, entre otros. El que esta metodología precise tanta diagramación para llevarse a cabo, supone que resulta difícil de aplicar si no se dispone de una herramienta CASE que facilite la labor [50].

### 3.4.3. Metodología de Wirfs-Brock

La metodología ‘responsabilidad-colaboración’, también conocida como R.D.D. (*Responsability Driven Design*) ha sido desarrollada por Rebeca Wirfs-Brock [187] y emplea como técnica CRC (*Class Responsability Collaboration*) o técnica de las tarjetas de clase.

Esta metodología consiste en la extracción de las clases del sistema a desarrollar y de la estructura jerárquica de las mismas, es decir, de sus superclases y de sus subclases. A partir de este punto se identifican las responsabilidades y colaboraciones de cada una. En la figura 3.2 se muestra una tarjeta de clase, donde se irán plasmando todos los elementos extraídos, teniendo en cuenta que, entre las ventajas que presenta la elaboración de las tarjetas, se encuentran:

- Las tarjetas son independientes de máquina y del lenguaje.
- Las tarjetas son intuitivas y baratas.

Resumiendo las fases por las que se ha de pasar si empleamos esta metodología son las siguientes:

- a) Fase 1<sup>a</sup>.- En ella se identifican clases y responsabilidades.
- b) Fase 2<sup>a</sup>.- Esta fase asigna responsabilidades a cada clase.
- c) Fase 3<sup>a</sup>.- Durante esta fase se identifican colaboraciones entre las diferentes clases existentes.

La metodología CRC/RDD es insuficiente por sí misma, pero facilita la extracción de las entidades del modelo y sus interacciones, pudiendo servir de gran ayuda en tareas de formación y como complemento de otras metodologías [50].

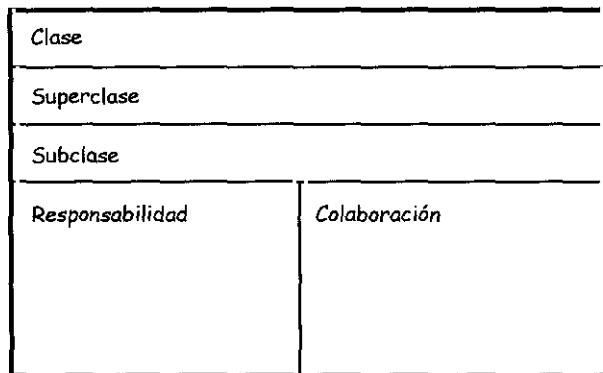


Figura 3.2. Tarjeta de clase en CRC/RDD

### 3.4.4. Metodología de Jacobson

La metodología O.O.S.E. (*Object-Oriented Software Engineering*), desarrollada por Jacobson (1992) emplea como técnica los casos de uso, también llamados formas de utilización (use cases). Los casos de uso han sido posteriormente incorporados a otras metodologías debido a la gran utilidad que presentan [92].

La metodología O.O.S.E. es realmente la versión reducida de Objetry. Consiste a grandes rasgos en describir un sistema desde el punto de vista del usuario y consta de las siguientes fases:

- a) Fase 1<sup>a</sup>.- En ella se muestran gráficamente las formas de utilización.
- b) Fase 2<sup>a</sup>.- Tiene como objetivo describir las formas de utilización mediante una definición clara de las necesidades del sistema.
- c) Fase 3<sup>a</sup>.- Durante esta fase se lleva a cabo la elaboración de un diagrama de clases.
- d) Fase 4<sup>a</sup>.- Esta fase establece las interacciones existentes en cada forma de utilización.

La metodología OOSE/Objectory es uno de los enfoques más industriales del desarrollo orientado a objetos [50].

### 3.4.5. Metodología de Rumbaugh

La técnica de modelado de objetos OMT (*Object Modelling Technique*) es una metodología de orientación a objetos [194] que cubre el análisis, diseño y programación informática de un sistema y que presenta la variedad más grande de conceptos de todas las metodologías hasta ahora descritas. Consiste en cuatro fases:

- a) Fase de análisis.- Su objetivo es la modelización del mundo real, haciendo uso de los modelos que se detallarán más adelante.
- b) Fase de diseño del sistema.- Durante esta etapa se agrupan en subsistemas los modelos obtenidos en la fase de análisis.
- c) Fase de diseño de objetos.- Consiste en definir cada objeto en detalle, lo que incluye definir su interfaz, algoritmos y operaciones. De nuevo los tres modelos se integran para diseñar los objetos.
- d) Fase de programación.- Su objetivo es traducir lo obtenido a un lenguaje de programación.

Según el autor, todo sistema tiene aspectos estáticos y dinámicos. El aspecto estático es fácilmente observable, mientras que el aspecto dinámico resulta más complejo de entender, ya que en él intervienen relaciones temporales. La evolución temporal de un sistema es lo que se denomina comportamiento [151]. El desarrollo de cada una de las fases anteriormente expuestas, se da lugar a la formación de tres modelos que van completándose de manera iterativa. Estos tres modelos son el Modelo de Objetos, el Modelo Dinámico y el Modelo Funcional. A continuación se expone en qué consiste cada uno de los modelos anteriormente enumerados.

### 3.4.5.1. Modelo de Objetos

El modelo de objetos describe la estructura estática de los objetos de un sistema y sus relaciones. Su objetivo es la elaboración de los diagramas de objetos, consistentes en un grafo cuyos nodos son las clases y cuyos arcos son las relaciones entre las clases. Los elementos que componen el modelo de objetos se detallan en la tabla 3.1. Asimismo se matiza que, en la construcción del modelo de objetos, se suceden los siguientes pasos:

- a) Identificar objetos y clases: Los objetos incluyen entidades físicas abstractas. Todas las clases deben tener sentido en el dominio de la aplicación, y no todas las clases son explícitas del problema, algunas son implícitas. Se pueden cometer errores en este primer paso, considerando clases a lo que realmente no lo son, por lo que habrá que eliminar clases redundantes, irrelevantes y difusas.
- b) Preparar un diccionario de datos: Un diccionario de datos comprende todas las entidades del modelo, junto con una pequeña descripción de las mismas.
- c) Identificar asociaciones (y agregaciones) entre objetos: Cualquier dependencia entre dos o más clases es una asociación. Una referencia de una clase a otra es una asociación. Se pueden cometer errores también en la identificación de las asociaciones, considerando como tales a las asociaciones irrelevantes, derivadas o las existentes entre clases eliminadas.
- d) Identificar atributos de los objetos: Los atributos son propiedades de los objetos individuales. Durante esta fase se procede a concretar los atributos de que constan las clases y objetos especificados en la primera fase del análisis.
- e) Organizar y simplificar las clases haciendo uso de la herencia: Se puede descubrir la herencia observando las clases desde arriba hacia abajo cuando tienen similares atributos, asociaciones u operaciones.
- f) Verificar que los accesos existen para las peticiones de datos: Siempre que se quiera obtener un dato, ha de existir la forma de conseguirlo.
- g) Iterar y refinar el modelo: Es necesario revisar varias veces el modelo, para asegurar su consistencia.
- h) Agrupar las clases en módulos: Un módulo es un conjunto de clases que comprende un subconjunto lógico de elementos del modelo. Este sería el momento de concretar los módulos de la aplicación si tuviera.

ELEMENTO	DESCRIPCION
<b>Clase</b>	Grupo de objetos con propiedades y comportamiento idénticos, relaciones comunes con otros objetos y semántica común. Una clase puede ser abstracta si no tiene objetos directamente.
<b>Atributo</b>	Característica que define a los objetos pertenecientes a una clase determinada.
<b>Operación</b>	Función o transformación que se puede aplicar a los objetos de una clase.
<b>Enlace</b>	Conexión física o conceptual entre las instancias de los objetos.
<b>Multiplicidad</b>	Relación que especifica cuántos objetos de una clase se pueden relacionar con un objeto único de una de las clases con la que esté asociada.
<b>Role</b>	Asociación específica entre dos clases.
<b>Agregación</b>	Asociación que representa una relación de composición.
<b>Herencia múltiple</b>	Herencia que detalla si un objeto hereda características de más de una clase.

Tabla 3.1. Elementos del Modelo de Objetos

### 3.4.5.2. Modelo Dinámico

El modelo dinámico describe los aspectos de un sistema que cambia a lo largo del tiempo. Se emplea para especificar y programar los aspectos de control de un sistema. El objetivo del modelo dinámico es la elaboración de los diagramas de estado, que consisten en grafos cuyos nodos son estados y cuyos arcos son transiciones causadas por eventos externos que ocasionan los cambios de estado. El modelo dinámico muestra el comportamiento de los objetos a lo largo del tiempo y sus elementos se representan en la tabla 3.2.

ELEMENTO	DESCRIPCION
<b>Evento</b>	Transmisión de información en una dirección concreta, de un objeto a otro, en un instante de tiempo. Un evento no tiene duración.
<b>Escenario</b>	Secuencia de eventos que ocurre durante la ejecución particular de un sistema. El ámbito de un escenario puede variar, puede incluir a todos los eventos de un sistema, o a aquéllos que generan ciertos objetos del sistema.
<b>Estado</b>	Abstracción de los valores de los atributos y de los enlaces de un objeto. El conjunto de valores se agrupa dentro de un estado de acuerdo a las propiedades que afectan al comportamiento del objeto.
<b>Diagrama de estados</b>	Diagrama que relaciona eventos y estados. Cuando se recibe un evento, el estado siguiente depende del estado actual así como del evento. Un cambio de estado causado por un evento se le llama <b>transición</b> . Un diagrama de estado es un grafo cuyos nodos son estados y cuyos arcos son transiciones entre los eventos.
<b>Condición</b>	Función booleana válida en un intervalo de tiempo. Un estado se puede definir en términos de una condición.
<b>Operación de Control</b>	Operación que gestiona el comportamiento del sistema. Puede ser de dos tipos: actividades y acciones. Una <b>actividad</b> es una operación que para finalizarla lleva su tiempo. La actividad está asociada a un estado. Una <b>acción</b> es una operación instantánea que se asocia con un evento y representa una operación cuya duración es insignificante, comparada con la resolución del diagrama de estados.
<b>Concurrencia</b>	Ocurrencia instantánea de dos eventos sin que exista interacción entre los receptores de los mismos.

Tabla 3.2. Elementos del Modelo Dinámico

La elaboración del modelo dinámico está determinada por las siguientes fases:

- Preparar los escenarios:** Un escenario es una secuencia de eventos. Un evento ocurre siempre que existe un trasiego de información entre un objeto del sistema y un agente externo al sistema. Los valores intercambiados constituyen los parámetros del evento.
- Identificar los eventos entre objetos:** Examinar los escenarios para identificar todos los eventos externos. Se pueden agrupar juntos los eventos que tienen el mismo efecto en el flujo de control, aunque sus parámetros difieran.

- c) Construir un diagrama de estados: Preparar un diagrama de estados para cada clase mostrando los eventos que el objeto envía y recibe; cada escenario, o secuencia de eventos, corresponde a un camino a través del diagrama de estados. Es importante destacar que no todas las clases tienen por qué disponer de un diagrama de estados.
- d) Validar eventos entre objetos: Chequear que los diagramas de estado, para cada clase, están completos y son consistentes.

### 3.4.5.3. Modelo Funcional

El modelo funcional describe las transformaciones en los valores de datos dentro del sistema; su objetivo es la creación de diagramas de flujo de datos, que son grafos cuyos nodos son procesos y cuyos arcos son datos. El modelo funcional muestra cómo son procesados los valores. En la tabla 3.3. se describen los elementos de que consta este modelo, en el que se siguen los siguientes pasos para la construcción del modelo funcional:

- a) Identificar los valores de entrada/salida: Los valores de entrada/salida son parámetros de los eventos entre el sistema y el mundo exterior.
- b) Construir los diagramas de flujo de datos: Un diagrama de flujo de datos se construye de acuerdo a un conjunto de niveles. En él se detallan las entradas y salidas y los almacenes de datos de los que se coge información o que se modifican.
- c) Describir las funciones: Una vez refinados los diagramas de flujo de datos, se escribe una descripción de cada una de las funciones, ya sea matemáticamente, en lenguaje natural o en pseudocódigo. Es necesario centrarse en lo que va a hacer la función.
- d) Identificar las restricciones: Las restricciones son dependencias funcionales entre objetos que no están relacionadas a una dependencia entrada/salida. Pueden actuar sobre dos objetos simultáneamente, entre instancias del mismo objeto o entre instancias de diferentes objetos.

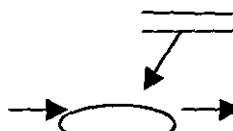
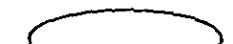
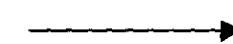
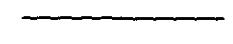
ELEMENTO	REPRESENTACION	DESCRIPCION
Diagrama de flujo de datos		Grafo que muestra un flujo de datos desde los objetos iniciales, a través de procesos que los transforman en objetos finales.
Proceso		Transformación de un valor en otro.
Flujo de datos		Conecta la salida de un objeto o proceso a la entrada de otro objeto o proceso.
Actor		Objeto activo que dirige el flujo de datos produciendo o consumiendo valores.
Almacén de datos		Objeto pasivo que almacena datos para un acceso posterior.

Tabla 3.3. Elementos del Modelo Funcional

### 3.4.6. Lenguaje unificado de modelado

El lenguaje de modelado unificado, *UML*, comienza su desarrollo en 1994, cuando Booch y Rumbaugh deciden desarrollar un estándar de modelado orientado a objeto, pero es en 1995 cuando aparece UML v.0.8. Posteriormente Jacobson se une aportando su metodología O.O.S.E. y creando en 1996 la versión 0.9 de U.M.L y en 1997 la versión 1.0, [18]. Las razones por las que se pensó en la posibilidad de elaborar un estándar para el análisis y diseño de un sistema fueron varias:

- Las tres metodologías que dieron lugar a UML (OOD; OMT; OOSE) estaban evolucionando por separado, creándose cada vez mayor distancia entre ellas, tanto en cuanto a semántica como en cuanto a notación.
- Se hacía necesario unificar la semántica y la notación en un estándar.
- Se iba a producir un proceso de retroalimentación entre los autores de las tres metodologías.

UML consiste en un metamodelo y en una notación. Decimos que es un metamodelo al facilitar todas las herramientas necesarias para describir formalmente los elementos de modelado y la sintaxis y la semántica de la notación que permite manipularlos [124]. Decimos que es una notación porque

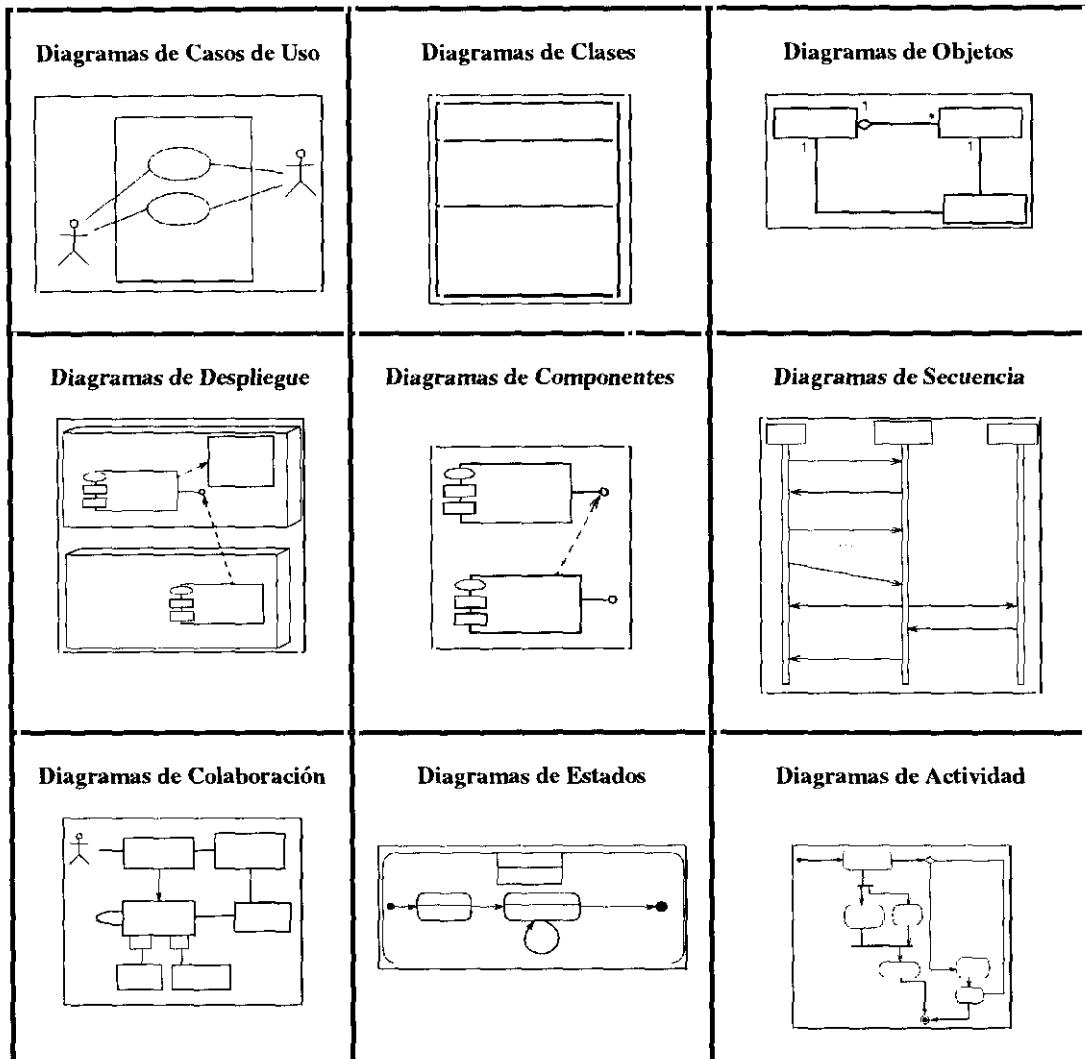
proporcionan los elementos necesarios para representar conceptual y funcionalmente un problema. UML surge, por tanto, como elemento unificador de todas las metodologías orientadas a objetos desarrolladas hasta el momento; de esta forma, al aparecer como estándar entre varias metodologías, presenta una diagramación mixta de todas ellas. UML dispone de dos tipos de formas de ver los datos de un sistema: desde un punto de vista estático y desde un punto de vista dinámico. El comportamiento particular que tienen los datos se engloban en vistas:

- Vistas estáticas.- Responden a una enumeración de las entidades de que consta el sistema. Estas vistas tiene como diagramas: Diagramas de Casos de Uso, Diagramas de Clases, Diagramas de Objetos, Diagramas de Despliegue y Diagramas de Componentes.
- Vistas dinámicas.- Responden a una descripción del comportamiento dinámico de las entidades anteriormente expuestas. Los diagramas de que constan son: Diagramas de Secuencia, Diagramas de Colaboración, Diagramas de Estados y Diagramas de Actividad.

Las vistas estáticas y dinámicas en UML contienen un conjunto de diagramas. A continuación describimos cada uno de los diagramas de que disponen y en la tabla 3.4 se puede observar un esquema gráfico de cada uno de ellos.

- Diagramas de Casos de Uso.- Representan la funcionalidad final del sistema. Los diagramas de casos de uso sirven para capturar los requisitos de un sistema y generar los casos de prueba. Estos diagramas derivan de la metodología O.O.S.E.
- Diagramas de Clases.- Representan las entidades del sistema y las colaboraciones entre ellas. Derivan de los diagramas de clases de OMT y de O.O.D.
- Diagramas de Objetos.- Representan los objetos del sistema.
- Diagramas de Despliegue.- Capturan la topología de un sistema hardware.
- Diagramas de Componentes.- Representan la estructura física de la programación informática; su objetivo es organizar el código fuente, constituir un ejecutable y especificar una base de datos.
- Diagramas de Secuencia.- Representan el comportamiento dinámico del sistema, teniendo en cuenta la dimensión temporal
- Diagramas de Colaboración.- Representan el comportamiento dinámico del sistema teniendo en cuenta los objetos que intervienen y las interacciones entre ellos.

- **Diagramas de Estados.**- Representan el comportamiento dinámico del sistema teniendo en cuenta los eventos existentes.
- **Diagramas de Actividad.**- Se trata de un caso especial de los diagramas de estados. Un diagrama de actividad se corresponde con una clase, una operación o un caso de uso.

Tabla 3.4. Diagramas propios de UML<sup>1</sup>

UML no es una notación propietaria, siendo accesible por todos de manera totalmente libre, es por lo tanto un estándar abierto. Por otra parte soporta el ciclo de vida de software completo y es suficientemente general como para ser aplicado a diferentes áreas, no sólo al ámbito informático.

<sup>1</sup> Diagramas recogidos de Booch, 1997.

## 3.5. Patrones orientados a objetos

### 3.5.1. Origen y fundamentos generales

El concepto de patrón surge en la década de los 70, cuando C. Alexander y sus colegas, arquitectos de profesión, observan que la construcción de edificios, vecindarios y ciudades se ajusta generalmente a unas pautas que se repiten bajo unas determinadas circunstancias. De esta forma publican dos libros [4, 5] en los que plantean el patrón como la descripción de una solución a un problema concreto que puede aplicarse tantas veces como se desee. Según el diccionario de la Real Academia un patrón es un ‘*modelo que sirve de muestra para sacar otra cosa igual*’. La idea de patrón por tanto no es específica de un dominio, puede aplicarse a cualquier área, sin embargo nosotros nos centraremos en el área informática que es la que nos ocupa.

La noción informática de patrón no está muy alejada de su definición básica. Ward Cunningham y Kent Beck, con motivo de la celebración de la *Conferencia para la programación orientada a objetos, sistemas, lenguajes y aplicaciones (OOPSLA)* en 1987, recuperaron la noción de patrón, aplicándolo al área informática. Su aportación principal fue la elaboración de cinco patrones destinados a programadores noveles de SMALLTALK con el fin de dar a conocer dicho lenguaje [87]. Resulta interesante el artículo de Beck y Cunningham publicado en el OOPSLA en 1991 donde establecen unas pautas para adentrarse en el mundo de los objetos [13]. A partir de este momento, se va desarrollando el concepto de patrón aplicado a la informática, como una relación entre tres elementos: un cierto contexto, unos requisitos que se repiten y una cierta configuración de software que permita resolver dichos requisitos.

Por otra parte, Erich Gamma, responsable en su mayor parte de la evolución de los patrones de diseño, de los que hablaremos más adelante, comienza su andadura en este tema, con la elaboración de su tesis doctoral sobre diseño orientado a objetos en ET++. Jim Coplien comienza a elaborar un catálogo de patrones específicos de C++, que él mismo denominó ‘*idiomas*’ y que han constituido un tipo de patrón, que sería publicado en 1991 [39]. Justo antes de la celebración de la *Conferencia Europea para la programación orientada a objeto (ECOOP)* en 1991, Gamma y Helm comienzan a pensar en la elaboración de catálogos de patrones, denominados más adelante ‘*patrones de diseño*’. En el OOPSLA de 1991, estos dos autores toman contacto con Johnson y Vlissides, y constituyen la llamada ‘Banda de los cuatro (GOF)’, escribiendo un libro que se publicaría más tarde [64] donde se catalogan 23 patrones de diseño: 5 de creación, 7 estructurales, y 11 de comportamiento. Esta obra viene a ser la referencia fundamental de muchos autores en cuanto a los patrones de diseño se refiere: tanto al concepto como a su catalogación. Según este grupo un patrón es una forma de comunicación entre clases y objetos cuyo propósito es resolver un diseño genérico que se da en un contexto

particular. Según Coplien y Schmidt [41] un patrón captura la estructura estática y dinámica de las soluciones que ocurren de forma repetitiva en aplicaciones bajo un determinado contexto.

En el desarrollo de aplicaciones, son muchas las veces que es necesario resolver los mismos tipos de problemas repetidas veces, para después implementarlos en diferentes sistemas informáticos. Por otro lado es común también que se dupliquen trozos de código de programa según surja la necesidad de emplearlos. Ambas alternativas conllevan a un mal uso de patrones (Coplien). Si se dispone de un código programado de forma genérica para emplearlo reiteradas veces, podemos afirmar que estamos utilizando la potencia de los patrones. El patrón consiste en abstraer y, por tanto, crear soluciones genéricas, que puedan ser compartidas en la misma aplicación o reutilizadas en distintas aplicaciones [147]; los patrones permiten la reutilización de la arquitectura y diseño de software, y capturan experiencia [155]. Por su utilidad práctica, el patrón parece imponerse en el área informática.

Según han ido avanzando las tecnologías, se han ido desarrollando diferentes tipos de patrones. Un ejemplo son los patrones denominados ‘de software’, iniciados por Beck y Booch en 1993, a quienes más adelante se les unirían Cunningham, Johnson, Auer, Hildebrand y Coplien, que junto con Gabriel planearían en 1994 llevar a cabo la primera conferencia de *Lenguajes de patrones para el diseño de programas (PloP)*, que se convertiría en realidad en 1995. El término ‘lenguaje de patrones’ ha sido introducido por Alexander para describir la pauta general que sigue la construcción de las ciudades. Según él, son los lenguajes de patrones los que proporcionan los elementos necesarios para combinar diferentes patrones y dar lugar finalmente a la creación de los edificios, jardines y en última instancia a las ciudades.

El lenguaje de patrones se propone pues, como un sistema que facilita al usuario la creación de patrones [4,5]. El lenguaje de patrón propuesto por Alexander constaba de más de 250 patrones, organizados desde un nivel superior a un nivel inferior. Si pensamos en el área informática, el lenguaje de patrones se puede plantear como un medio muy eficaz para documentar arquitecturas de software [14], aunque no se debe confundir con una metodología de diseño. La diferencia estriba en que escribir un patrón forma parte de la experiencia y se propone como solución genérica que otros desarrolladores deben poder usar si se están enfrentando a un problema de similares características.

Los primeros patrones de software estuvieron dirigidos a desarrolladores O.O, y se centraron fundamentalmente en el diseño y la programación O.O. [64] mediante modelado O.O [35]. Además, las publicaciones y descubrimientos de patrones han tratado siempre de resolver problemas en este paradigma, no significando que estén asociados por definición a metodologías o a lenguajes (Rising, 1999). Por ejemplo se han desarrollado también patrones que no presentan relación alguna con el paradigma O.O. como 4ESS [2]. Coplien [40] ha tratado de recopilar aquellas preguntas que más comúnmente uno se plantea ante la aparición del patrón aplicado a diseño de software. Este trabajo presenta cuestiones tan interesantes como la diferencia existente entre paradigma y patrón, o el porqué un patrón no se debe considerar una simple regla de funcionamiento. La formalización de patrones también ha sido tema sugerente para escribir normas acerca de cómo llevarla a cabo, Vlissides en su

documento [174] proporciona una serie de hábitos que deben tenerse para escribir con éxito los patrones.

Tradicionalmente la ingeniería de software conlleva todo el proceso de recogida de requisitos, la formalización del problema empleando cualquier tipo de metodología y la programación de la herramienta. Actualmente la noción de ingeniería se ha sustituido por la de arquitectura, poniendo énfasis no tanto en las necesidades iniciales del sistema, sino en la construcción de componentes que se integren fácilmente unos con otros y desemboquen en la elaboración del sistema. De esta forma se involucra al usuario durante prácticamente todo el tiempo que dura el desarrollo de la herramienta en contraste con el enfoque tradicional, en que el usuario exponía sus necesidades al analista y ya no intervenía en ninguna de las fases posteriores del desarrollo.

Resumiendo, los patrones se caracterizan por:

- Separar interfaz de implementación.
- Separar lo que es común de lo que es variable.

El uso de patrones, durante las tareas de elaboración de una aplicación presenta varias ventajas, tales como permitir la reutilización de arquitecturas de software de gran escala, capturar el conocimiento experto, ayudar a incrementar la comunicación del desarrollador y hacer fácil la migración a la tecnología de objetos.

### 3.5.2. Clasificación de patrones

Los patrones se pueden clasificar atendiendo a su nivel de abstracción, en [95] se puede consultar una clasificación de patrones de diseño orientados a objetos bastante completa, nosotros expondremos los patrones que nos resultan más interesantes por la relación que guardan con el presente trabajo:

- Patrones de arquitectura.- Expresan una organización estructural para los sistemas de software. Los patrones de arquitectura proporcionan un conjunto de subsistemas, especificando las responsabilidades de cada uno e incorporando reglas para relacionarlos entre sí.
- Patrones de diseño.- Se plantean en un nivel intermedio, entre las clases y objetos y la aplicación final; es en este nivel donde surgen las decisiones de diseño propiamente dichas. Este tipo particular de patrones proporciona una solución genérica que puede aplicarse a problemas particulares.

- Patrones de codificación o idiomas.- Relativos a la implementación de un aspecto particular en un determinado lenguaje de programación.
- Marcos (frameworks).- Los marcos son algo más que un patrón genérico, ya que constituyen código de programación. Consisten en un conjunto integrado de componentes que colaboran para proporcionar una arquitectura reutilizable en diversas aplicaciones. La documentación de marcos puede hacerse mediante patrones, tal y como especifica Johnson en su trabajo de 1992 [94]. Por ejemplo un marco puede estar constituido por un conjunto de clases abstractas y la forma en que instancias de dichas clases interaccionan entre sí. Algunos ejemplos destacados son: MFC (Microsoft Foundation Classes) de Microsoft ó Broker [23]. Los marcos permiten la reutilización del código y del diseño específico si se emplea la potencia de la orientación a objetos en su construcción [197] y tanto patrones como marcos contribuyen al desarrollo de software de calidad, reduciendo el tiempo de desarrollo; desarrollar un buen marco suele resultar bastante caro, ya que interviene la experiencia no sólo de una persona sino de varias, que deben colaborar para culminar con éxito la elaboración de marcos.

La descripción de un patrón de diseño es independiente del lenguaje de programación o detalles de implementación, representando, por tanto, un nivel de abstracción mayor que los propios marcos.

Si atendemos a la agrupación de patrones entre sí, se consideran los siguientes:

- Catálogos de patrones.- Colección de patrones relacionados entre sí.
- Sistemas de patrones.- Colección de patrones que interactúan entre sí para proporcionar la construcción y evolución de arquitecturas.

Otra forma de clasificar los patrones es subdividiéndolos en: patrones conceptuales, de diseño y de programación, guardando relación respectivamente con el análisis, diseño e implementación de todo desarrollo informático [145].

### 3.5.3. Lenguajes de patrones

Un lenguaje de patrones es un conjunto de patrones que se usan conjuntamente para resolver un determinado problema. Los lenguajes de patrones surgen a partir de dos necesidades muy concretas; por un lado como una forma de comprender y posiblemente controlar un sistema complejo; por otro como herramienta de diseño con la cual construir algo plenamente funcional y estructuralmente coherente.

Según Salingaros [153] un lenguaje de patrones, para considerarse como tal, ha de añadir en su estructura tanto un cierto factor de escala como un nivel jerárquico; esto facilita la validación del mismo, puesto que en esta fase se trata de ir agrupando los patrones más concretos para alcanzar los más generales, de forma que van renaciendo propiedades nuevas que los patrones por separado no tenían. Siguiendo con el mismo autor, éste considera los patrones componentes de un lenguaje como un grafo donde los nodos son dichos patrones, y las aristas que los unen representan las reglas por las que deben encajar unos con otros. Análogamente a lo que ocurre en los sistemas biológicos, un sistema informático se comporta adecuadamente porque las conexiones entre los subsistemas que lo forman, funcionan de manera correcta [131]. Entre los lenguajes de patrones desarrollados, no podemos olvidar el precursor de los mismos: el propuesto por Alexander [4] constituido por más de 250 patrones relacionados con la arquitectura.

A veces se desarrollan lenguajes de patrones que sirven para desarrollar tareas relacionadas a su vez con patrones que proporciona las pautas necesarias para llevar a cabo el desarrollo de marcos orientados a objetos [148], o por ejemplo el propuesto en Meszaros [120] que facilita recomendaciones para la formalización de los patrones, mediante la elaboración de un lenguaje de patrones.

Otros lenguajes de patrones están relacionados directamente con labores de análisis y programación denominado 'RAPPeL' que guía a jefes de proyecto, analistas y desarrolladores para determinar y definir las necesidades para el desarrollo de aplicaciones de negocio que se quieran apoyar en un entorno orientado a objetos [184]; el lenguaje de patrones 'METAMORPHOSIS' presenta como objetivo proporcionar un lenguaje de patrones para que haga de guía en la transformación de un análisis a un diseño [97]. El lenguaje de patrones 'EPISODES' describe cómo desarrollar software adecuado en cuanto a temas comerciales se refiere [45].

### 3.5.4. Patrones de diseño

Los patrones de diseño capturan la estructura estática y dinámica y la colaboración entre los elementos involucrados durante el diseño de software, proporcionando un vocabulario común. Los patrones de diseño se clasifican atendiendo tanto a su caracterización como a su jurisdicción [63].

La caracterización de los patrones de diseño establece tres tipos de patrones:

- Patrones de creación.- Se relacionan con la inicialización y configuración de clases y objetos. Un ejemplo es 'Abstract Factory' [63].
- Patrones estructurales.- Tratan de la separación de la interfaz y la implementación de las clases y objetos. Un ejemplo es 'Wrapper' [63].

- Patrones de comportamiento.- Describen las interacciones existentes entre las clases y los objetos. Un ejemplo es ‘Object Behavioral’[63].

La jurisdicción de un patrón de diseño viene dada por el ámbito de las relaciones. Los patrones de diseño pueden tener ámbito de clase, si las relaciones ocurren entre clases y subclases; ámbito de objeto, si ocurren entre objetos o ámbito de compuesto si ocurren entre objetos que presentan cierta estructura recursiva.

En la última década se han propuesto diferentes formas de documentar un patrón de diseño O.O. Quizá la que más se ha impuesto ha sido la de Gamma [64], según quien, todo patrón de diseño tiene una serie de elementos esenciales:

- Nombre.- El nombre del patrón debe ser lo suficientemente significativo que indique el problema a que se va a aplicar y la solución que ofrece.
- Objetivo.- Se trata de describir para qué tipo de problemas es adecuado.
- Motivación.- Con este elemento se pretende presentar qué es lo que ha movido al autor a crear dicho patrón, puede ser una breve introducción sobre cómo se resuelve el problema normalmente sin hacer uso del patrón y cómo se puede resolver haciendo uso del mismo.
- Aplicación.- Es necesario explicar la utilidad práctica resultante de usar el patrón.
- Participantes.- Se trata de hacer una descripción de los objetos y clases que intervienen en el patrón.
- Colaboraciones.- Establece cómo colaboran entre sí los participantes anteriormente descritos para llevar a cabo las responsabilidades para las que fueron creados.
- Diagrama.- Se trata de una representación gráfica del patrón usando la notación OMT.
- Consecuencias.- Este elemento determina las implicaciones que se derivan una vez que el patrón se ha aplicado a un problema concreto.
- Implementación.- Consiste en especificar el posible lenguaje de programación con que se podría implementar dicho patrón.
- Ejemplos.- Se trata de enumerar ejemplos de desarrollos en que se haya empleado el patrón.

El futuro de los patrones diseño ya era adivinado por Schmidt [155], cuando aseguraba que una aplicación interesante sería la integración de dichos patrones para formar tanto marcos como lenguajes de patrones. Fue el mismo Schmidt quien desarrolló el patrón de diseño denominado *Wrapper Facade*, aplicable a la construcción de marcos [156]. Se han desarrollado herramientas para la generación automática de patrones de diseño, que, a partir de datos muy concretos del patrón, la herramienta genera las declaraciones y definiciones de las clases, para que el usuario tan sólo tenga que integrar este código con el resto de la aplicación [22].



# **ELABORACIÓN DE UN LENGUAJE DE PATRONES**

**FUNDAMENTOS GENERALES  
 LENGUAJE DE PATRONES PARA LA SÍNTESIS Y  
 REGULACIÓN DE OBJETOS VISUALES**

# **Capítulo 4**

---

## **Fundamentos generales**

### **4.1. Patrón evolutivo de la naturaleza**

El mecanismo de evolución de la naturaleza basa sus principios en la transferencia de la información genética de unos organismos a otros, dando lugar a la construcción de un patrón genérico que puede servir de molde, durante la realización de aquellas tareas cuyo objetivo sea la transformación de unidades sencillas de información en unidades más complejas. De esta forma utilizaremos la biosíntesis de proteínas como medio para proponer un lenguaje generador de patrones de diseño que abarque el recorrido que toda unidad de información realiza hasta que puede ser utilizada directamente.

De esta forma genérica, pasamos a describir cada uno de los elementos que intervienen en el patrón evolutivo que proponemos, señalando que nos parece más formativo ir planteando el funcionamiento del mismo en forma de preguntas, con el fin de que a cada uno de los elementos que vayan apareciendo podamos fácilmente asociarle la función específica que tienen. Posteriormente, en el capítulo 5, cuando mostremos la estructura interna del lenguaje de patrones propuesto, lo plantearemos en los mismos términos: mediante preguntas, con el fin de que resulte más sencillo

analizar cómo ALBA emplea como molde el patrón evolutivo que proporciona la biosíntesis de proteínas.

La primera pregunta debe ser esencialmente informativa:

**¿Dónde reside la información genética?**

El ácido desoxirribonucleico, *a partir de ahora ADN*, residente en el núcleo celular, es el portador directo de la información genética; esta molécula presenta un modelo estructural basado en una doble hélice enrollada alrededor del eje [182]. El ADN pues, está constituido por dos cadenas helicoidales que se mantienen unidas por las bases púricas y pirimidínicas, cuyos planos son perpendiculares al eje. Esta unión se hace por enlaces de hidrógeno entre pares de bases, una de cada cadena y una de cada tipo, entre las posiciones 1 y 1 y 6 y 6 de las bases púricas y pirimidínicas, respectivamente. El apareamiento entre bases es específico: adenina con timina y guanina con citosina.

**¿Quiénes son los últimos receptores de la información genética?, es decir,  
¿Cuándo dicha información se convierte en conocimiento? [191]**

La hipótesis de la secuencia [42] establece una ordenación lineal entre los nucleótidos de los ácidos nucleicos y los aminoácidos de las proteínas [154]. La molécula de ADN es una molécula lineal codificada en forma de una secuencia específica de cuatro nucleótidos básicos, que contiene instrucciones para la estructura y función biológicas. Estas instrucciones se llevan a cabo por las proteínas que constituyen también un lenguaje altamente específico y que a diferencia de la estructura lineal del ADN, presenta una estructura tridimensional estable.

Se establecen dos principios básicos relacionados con el ADN y los organismos:

- La cantidad de ADN de una especie, célula u organismo determinado es constante y no puede ser alterada.
- La cantidad de ADN por célula es proporcional a ésta y por tanto a la cantidad de información genética que contiene. Cuanto más alto se encuentra un organismo en la escala evolutiva, mayor es el contenido de ADN por célula.

Partimos, pues, de que la información genética está contenida en el ADN, que es la macromolécula informativa de los cromosomas, cuya existencia se remonta billones de años [43].

La secuencia aminoacídica de la cadena polipeptídica de cada tipo de proteína es la que resulta codificada por la secuencia de nucleótidos en el ADN. El segmento de una molécula de ADN, que especifica una cadena polipeptídica completa, es lo que se denomina gen. Cada aminoácido está

codificado por tres restos nucleotídicos sucesivos en el ADN o triplete codificador. Durante esta fase los genes permanecen en los cromosomas y no actúan directamente como matrices de codificación durante la biosíntesis de proteínas, que tiene lugar en los ribosomas.

**¿Quién se encarga de transportar la información genética residente en los genes desde el núcleo hasta el citoplasma?**

El mensaje genético del gen se transmite primero enzimáticamente para formar un tipo específico de ácido ribonucleico llamado ARN mensajero, *a partir de ahora ARNm*, proporcionando a continuación la información genética que especifica la secuencia de los aminoácidos durante la biosíntesis de proteínas. El ARNm es la copia activa de la información genética, en él se incorporan las instrucciones codificadas en el ADN, para que pueda dictar la secuencia de aminoácidos a las proteínas. Ante toda esta secuencia de instrucciones para llevar a cabo la transmisión del mensaje genético, surge la siguiente pregunta:

**¿Cómo se traduce la información genética contenida en el ARNm a la estructura proteica?**

Todo el proceso de la transmisión de la información genética constituye el dogma central de la biología molecular, enunciado por Crick en 1970, como se expone en la figura 4.1.

$$\text{ADN} \rightleftharpoons \text{ARNm} \rightarrow \text{Proteína}$$

Figura 4.1. - Dogma central de la biología molecular

Los principios básicos del dogma central de la biología molecular son los siguientes:

- Replicación semiconservativa del ADN.**- La replicación semiconservativa del ADN transmite instrucciones de la célula madre a las células hijas y de generación en generación. Este proceso está catalizado por la ADN-polimerasa que replica y repara el ADN.
- Transcripción.** - Cada molécula de ARNm se copia o transcribe de una de las dos cadenas de ADN, que actúa de cadena molde, con el fin de llevar el mensaje genético a los ribosomas. Este proceso es catalizado por la enzima ARN-polimerasa.
- Traducción.** - Inicialmente los aminoácidos se encuentran dispersos en el citoplasma. La unión de los aminoácidos mediante enlaces polipeptídicos y siguiendo un orden concreto, es lo que constituye cada uno de los diferentes tipos de proteínas. La secuencia lineal de los aminoácidos en las proteínas es lo que determina su especificidad, y es lo que se ha transmitido desde el ADN hasta el ARNm. El paso de la información genética desde el

ARNm hasta las proteínas propiamente dichas es lo que se determina traducción y en ella juega un papel importante el ARN transferente, *a partir de ahora ARNt*, que actúa como adaptador molecular con el fin de aceptar el aminoácido para poder ser ajustado al lenguaje de tripletes nucleotídicos del código genético.

En la figura 4.2. exponemos un esquema gráfico del funcionamiento de la biosíntesis de proteínas en la célula, viene a ser la expresión gráfica del dogma central de la biología molecular.

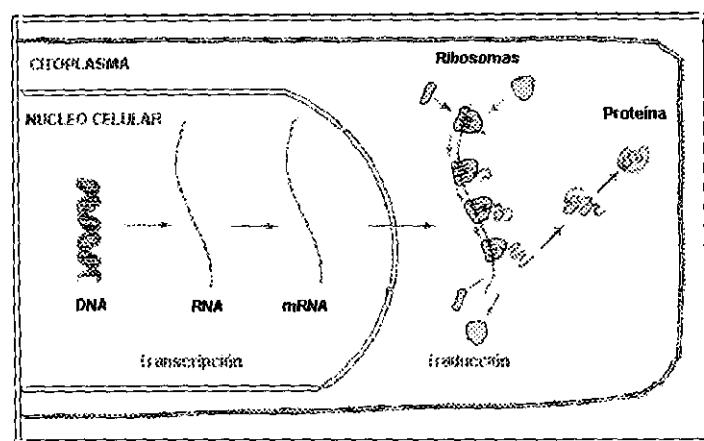


Figura 4.2. Proceso de la biosíntesis de proteínas

La traducción, a su vez, consta de cuatro etapas: activación, iniciación, prolongación y terminación, tal y como ilustramos en la figura 4.3.

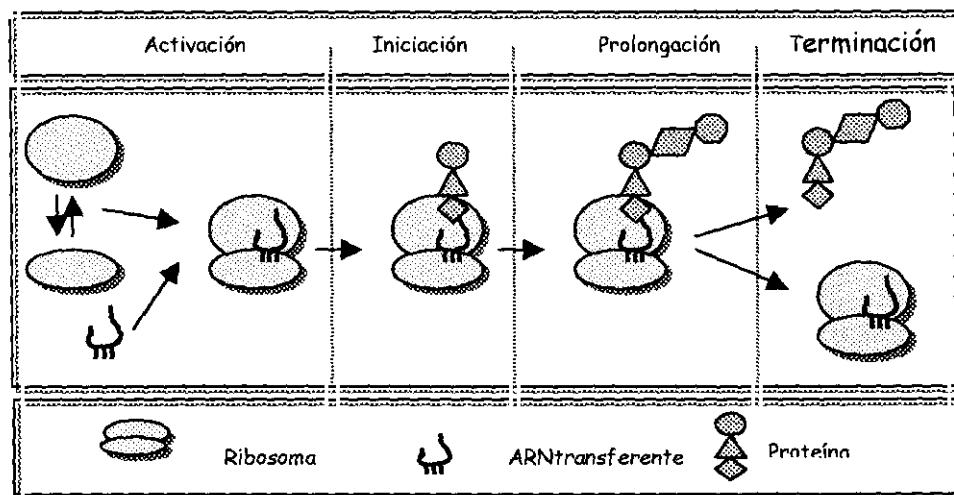


Figura 4.3. Vista del proceso de traducción de la biosíntesis de proteínas

A continuación pasamos a describir cada una de las etapas anteriormente enunciadas:

- a) Activación.- Los aminoácidos se unen mediante un proceso energético a los ARNt, actuando como enzima la aminoacil-ARNt sintetasa.
- b) Iniciación.- El complejo formado por el primer aminoácido, el ARNt, y a su vez el ARNm se unen a la subunidad menor del ribosoma. A continuación la subunidad mayor se adhiere entonces para formar un ribosoma funcional listo para la siguiente etapa.
- c) Prolongación.- La cadena polipeptídica se va prolongando, por adición sucesiva de complejos aminoácido-ARNt, generándose poco a poco los complejos proteicos.
- d) Terminación.- Cuando la cadena polipeptídica se ha completado, entonces por acción de unas proteínas denominadas factores liberadores, el producto se libera del ribosoma.

La proteína es pues el sueño del gen hecho realidad [85]. Una vez completado el proceso de traducción, cada proteína recién sintetizada posee una forma tridimensional característica, determinada por los niveles estructurales que ilustramos en la figura 4.4. y que detailaremos a continuación:

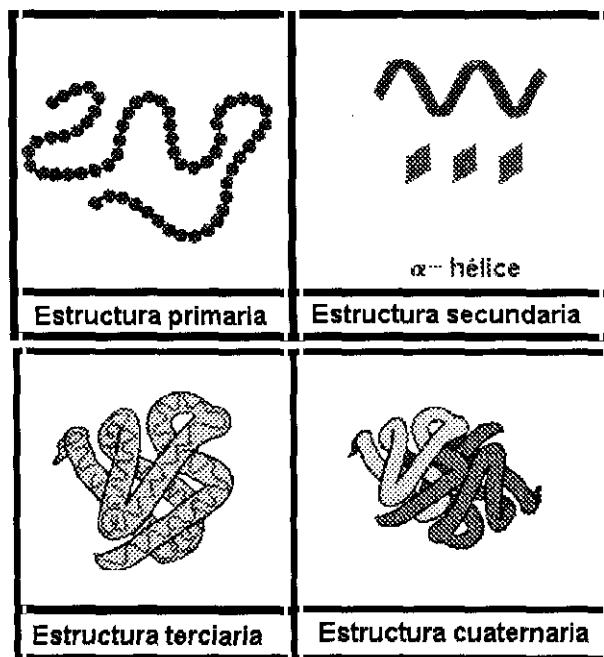


Figura 4.4. Niveles de la estructura proteica

- a) Nivel primario.- Viene dado por la identidad, cantidad y secuencia de los aminoácidos que forman las cadenas polipeptídicas.
- b) Nivel secundario.- Se refiere a la orientación de las proteínas, estabilizadas por los puentes de hidrógeno.
- c) Nivel terciario.- Determina la estructura tridimensional de las proteínas, es decir, el modo en que la cadena polipeptídica se pliega para formar estructuras compactas.
- d) Nivel cuaternario.- Se refiere a la disposición en el espacio de varias cadenas polipeptídicas de una proteína cuando está formada por más de una cadena.

Las proteínas son las moléculas orgánicas más abundantes en las células y son fundamentales tanto en el ámbito de la estructura como de la función celular. Existen muchas clases de proteínas diferentes, cada una de ellas especializada en una función biológica distinta. Entre ellas cabe destacar:

- a) Proteínas transportadoras.- Trasladan diferentes componentes de un sitio a otro, por ejemplo la hemoglobina, que transporta oxígeno en la sangre de los vertebrados o la mioglobina, que lo transporta en el músculo.
- b) Hormonas o Proteínas reguladoras.- Llevan a cabo la regulación de diferentes mecanismos biológicos que ocurren en la célula. Por ejemplo la insulina, que regula el metabolismo de la glucosa o la hormona del crecimiento, que regula el crecimiento de los huesos.
- c) Proteínas estructurales.- Llevan a cabo el soporte celular de diferentes tipos de macromoléculas como el colágeno que forma parte del tejido conectivo fibroso o la elastina, que constituye el tejido conectivo elástico.
- d) Enzimas.- Intervienen como activadores en las reacciones. Destaca por ejemplo la ADN-polimerasa que replica y repara el ADN, o el citocromo-c que realiza la transferencia de electrones en el ámbito celular.
- e) Proteínas defensivas.- Llevan a cabo una función de defensa en el organismo, destacan por ejemplo anticuerpos como las inmunoglobulinas, la trombina y el fibrinógeno que evitan la formación de hemorragias mediante la creación de coágulos sanguíneos.

Por tanto, la existencia de una proteína viene respaldada por la acción que va a ejercer en el organismo.

## 4.2. Presentación del lenguaje de patrones

Al igual que la naturaleza se conserva a sí misma haciendo uso de los mecanismos evolutivos y de transmisión de la información genética de que dispone, los sistemas informáticos se mantienen vivos si existe un mecanismo apropiado de mantenimiento tanto de la información que manejan, como de su propia arquitectura. La primera afirmación se consigue si se ha realizado un diseño consistente de la base de datos que por lo tanto, haga sencilla su modificación. La segunda se obtiene si el sistema dispone de una buena documentación de diseño.

Actualmente los principios de la O.O. promueven tanto el uso de una terminología común entre informáticos y usuarios como la construcción de aplicaciones modulares fáciles de mantener. Por otra parte, la programación conducida por eventos, fuerza al desarrollador de aplicaciones a dividir el programa en pequeñas funciones y a reducir el ámbito de las variables únicamente a la función que las maneja [137]; pero si no existe una documentación de apoyo que siga paso a paso cada una de las entidades y funciones de la aplicación, el mantenimiento seguirá resultando un problema. Existe una gran variedad de metodologías y patrones que facilitan las tareas de desarrollo de una aplicación, pero con frecuencia o bien están basadas en una serie de etapas poco intuitivas de manera inmediata, en el caso de las metodologías, o bien ofrecen soluciones parciales a problemas muy concretos pero adolecen de una solución integral a la resolución de un problema informático, desde la fase de almacenamiento físico de los datos hasta lo que constituye la fase de construcción de la interfaz de usuario.

El desarrollo de las aplicaciones informáticas lo planteamos pues, como un proceso evolutivo, que recorre una serie de etapas que van desde el almacenamiento críptico de la información hasta el manejo de la misma por parte del usuario. Por esta razón proponemos un lenguaje de patrones denominado ‘Lenguaje de patrones para la Síntesis y Regulación de Objetos Visuales (ALBA)’ que facilita la construcción de patrones de diseño genéricos y cuya elaboración se ha inspirado en el patrón evolutivo que la biosíntesis de proteínas proporciona.

Por otro lado hemos observado en la naturaleza una relación directa entre funcionalidad y regiones celulares, orgánulos u órganos. Así, al igual que las mitocondrias se relacionan directamente con la respiración o el núcleo celular con el almacenamiento de la información genética, en el lenguaje de patrones ALBA también contemplamos una cierta compartimentalización funcional, basando su estructura en un modelo de tres capas. Así cada capa tiene asignada una funcionalidad concreta que explicaremos con más detalle a continuación.

### 4.2.1. Nombre

El nombre elegido para el lenguaje de patrones que proponemos ha sido: **Lenguaje de patrones para la Síntesis y Regulación de Objetos Visuales**, abreviadamente ALBA, por su acepción: ‘Amanecer o empezar a aparecer la luz del día’, debido a que consideramos este lenguaje por un lado como un generador de patrones de diseño específicos y por otro como un mecanismo para crear objetos visuales que originan la **aparición** de una información empaquetada crípticamente. A continuación explicamos el significado del nombre más ampliamente:

- **Lenguaje.**- Formalización de un conjunto de signos y símbolos, que responden a una notación concreta y cuyo objetivo es facilitar la comunicación.
- **Patrón.**- Descripción de una solución probada cuyo objetivo es proporcionar las etapas y recursos necesarios para la realización de una determinada tarea.
- **Síntesis.**- Proceso que permite obtener sustancias a partir de sus componentes.<sup>2</sup> Proceso de construcción de clases concretas que tienen una responsabilidad determinada en el sistema.<sup>3</sup>
- **Regulación.**- Proceso de ajuste del funcionamiento de un sistema a determinados fines.<sup>2</sup> Proceso de formalización del comportamiento de cada una de las clases elaboradas durante la síntesis. La regulación plasma cómo las diferentes clases colaboran entre sí para llevar a cabo las funciones para las que han sido creadas.<sup>3</sup>
- **Objetos Visuales.**- Pantallas que en última instancia va a manejar el usuario. De forma genérica son las clases visuales, que cuando cada una de sus propiedades adquiere valores, pasan a convertirse en objetos.

### 4.2.2. Objetivo

El lenguaje de patrones propuesto presenta como objetivo facilitar la elaboración del análisis y diseño informáticos basándose en la interacción responsabilidad - colaboración que subyace en todo problema a resolver, sea cual sea su naturaleza. De esta forma este lenguaje se plantea como un todo integrado por cuatro patrones de diseño genéricos, denominados modelos por su paralelismo con el patrón evolutivo antes mencionado. Cada uno de dichos patrones ocupa un nivel diferente en una escala jerárquica imaginaria, interactuando entre sí, y dispone de un objetivo concreto que le convierte en pieza indispensable para culminar con éxito el desarrollo de una aplicación informática.

---

<sup>2</sup> según el diccionario de la Real Academia Española

<sup>3</sup> según el patrón propuesto

### 4.2.3. Motivación

Hemos observado en la naturaleza mecanismos biológicos cuyo objetivo y posterior desarrollo no difieren de los mecanismos que se desencadenan cuando se está diseñando un sistema informático. Es más, surgen unas necesidades comunes en ambos planteamientos y como consecuencia una forma paralela de solucionarlas. Así, concluimos que de todos los fenómenos existentes en la naturaleza, la biosíntesis de proteínas proporciona un patrón evolutivo perfecto a imitación del cual puede llevarse a cabo la construcción de determinados sistemas informáticos, tal y como se muestra en la figura 4.5.

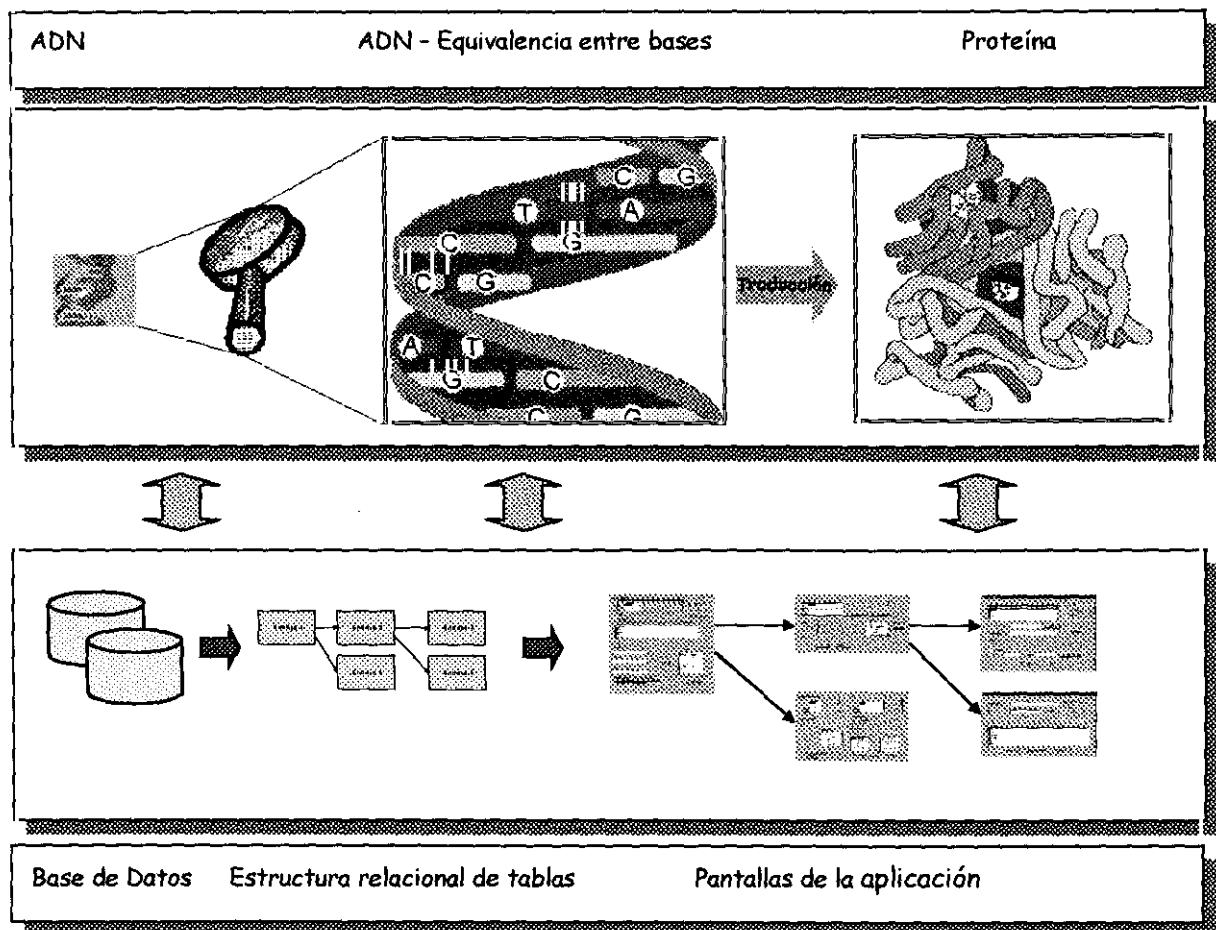


Figura 4.5. Comparación de la biosíntesis de proteínas con ALBA

Analizando objetivos en uno y en otro caso, encontramos que el objetivo fundamental de la biosíntesis de proteínas es hacer visible y operativa una información previamente empaquetada. Análogamente, uno de los objetivos de la construcción de un sistema informático es hacer visible y manipulable una información previamente conocida. Por esta razón, y otras que iremos desvelando a

medida que vayamos adentrándonos en el lenguaje propuesto, estimamos adecuado proponer el patrón evolutivo subyacente en la biosíntesis de proteínas, como molde del lenguaje de patrones para la síntesis y regulación de objetos visuales.

En la figura 4.5, tal y como decíamos anteriormente, ilustramos una comparación gráfica de lo que ocurre en el patrón evolutivo y en el lenguaje, a diferente nivel de observación. En dicha figura establecemos a un nivel más interno, una analogía entre el almacenamiento de bases en el ADN y el almacenamiento de información en bases de datos. A continuación ascendiendo en el nivel de detalle, se observan las bases púricas y pirimidínicas unidas mediante enlaces hidrofóbicos, del mismo modo, la información se agrupa en tablas, guardando una estructura relacional entre ellas. A un nivel más superficial, mostramos ya la estructura espacial de una proteína, que guarda semejanza con las salidas de cualquier sistema informático, con sus pantallas de datos relacionadas unas con otras.

Por todo ello ALBA tiene la particularidad de plasmar de una forma gráfica, mediante su propia notación, la semántica que le da sentido, proporcionando una serie de etapas donde semántica y notación están íntimamente unidas y no puede explicarse la una sin la otra.

#### **4.2.4. Aplicación**

El lenguaje de patrones propuesto es ideal para desarrollar aplicaciones informáticas que cumplan tres requisitos: por un lado han de disponer de una cierta información susceptible de ser almacenada, por otro lado deben requerir de un nivel de manipulación de los datos almacenados y por último deben precisar de una interfaz de usuario que muestre dicha información. Resumiendo: ALBA se aplica a desarrollos de herramientas basados en modelos de tres capas: almacenamiento, gestión y visualización.

#### **4.2.5. Estructura**

La estructura básica de ALBA gira en torno a dos ideas: la responsabilidad que tiene cada entidad en sí misma, entendiendo por entidad cualquier unidad conceptual de información, y la colaboración que ha de existir entre ellas con el fin de cubrir la funcionalidad para la que fueron creadas. De esta forma, basándonos en estas dos ideas, establecemos un lenguaje de patrones compuesto por dos etapas: Responsabilidad y Colaboración, las cuales se estructuran a su vez en patrones o modelos. El motivo de esta composición es diferenciar la generación de patrones de diseño cuyo objetivo sea conceptual de aquéllos que cubran objetivos funcionales.

Proponemos pues, cuatro patrones de diseño que hemos dado en llamar modelos por aproximarnos más, semánticamente hablando, a lo sucedido en los procesos biológicos en que nos

basamos. Cada uno de estos modelos ofrece una descripción del sistema desde una particular perspectiva. Los dos primeros cubren una responsabilidad conceptual y funcional, respectivamente, al extraer todas las entidades del sistema y detallar todos sus aspectos funcionales. Los dos siguientes patrones que proponemos cubren la colaboración conceptual y funcional, respectivamente, que consiste en detallar cómo las entidades interaccionan unas con otras para cubrir la funcionalidad del sistema. El éxito del lenguaje se basa en que los cuatro patrones a que da lugar, encajan perfectamente como las piezas de un puzzle, para llevar a cabo el desarrollo de una herramienta informática.

A continuación detallamos el objetivo que ha de cubrir cada uno de los patrones de diseño o modelos del lenguaje propuesto:

- a) Patrón o Modelo de Síntesis de Objetos Visuales (PASOV).- Mediante él se detalla todo el conjunto de clases que intervienen en el sistema a diseñar. Su objetivo, por tanto, es aportar una dimensión conceptual a la aplicación, es decir, cubrir la responsabilidad conceptual.
- b) Patrón o Modelo de Regulación de Objetos Visuales (PAROV).- Es el encargado de concretar la funcionalidad del sistema, haciendo referencia a cada una de las clases anteriormente extraídas. Su objetivo es plasmar la responsabilidad funcional en la aplicación.
- c) Patrón o Refinamiento del Modelo de Síntesis de Objetos Visuales (PARSOV).- Ilustra la navegación a través de las pantallas que va a existir en la aplicación. Su objetivo consiste en mostrar la colaboración conceptual entre clases.
- d) Patrón o Refinamiento del Modelo de Regulación de Objetos Visuales (PARROV).- Se emplea para concretar cómo las clases colaboran entre sí para cubrir la funcionalidad para la que han sido creadas. Su objetivo es plasmar la colaboración funcional.

Todos estos objetivos pueden resumirse más concretamente en los siguientes:

- Separar lo esencial del detalle,
- Separar los conceptos de la implementación y
- Emplear una terminología intuitiva.

Estos objetivos se consiguen mediante la consecución de una serie de fases dentro de cada modelo. Cada una de estas fases están constituidas a su vez por una serie de vistas que proporcionan una representación parcial del sistema y que son semánticamente consistentes entre sí. Las vistas son representaciones gráficas mediante diagramas cuya notación se basa esencialmente en OMT, aunque hay algunas propuestas de diagramas realizadas por la autora.

A continuación se detallan las etapas de que consta ALBA y en la figura 4.6. se muestran cada uno de los modelos que aparecen y las fases de cada uno de ellos. La notación empleada, como ya decímos es prácticamente la aportada en OMT, habiéndose necesitado incluir nuevos diagramas para determinadas fases del lenguaje. En las figuras que van desde la 4.7 hasta la 4.10, se ilustra la notación genérica empleada para cada una de las fases de los patrones o modelos y en los apéndices A y B del anexo se especifica más en detalle tanto la notación gráfica como la notación alfanumérica respectivamente.

En el capítulo siguiente nos centraremos en cada uno de los patrones de diseño por separado. La definición y caracterización de cada uno de ellos la hacemos empleando los criterios que hemos encontrado en Gamma [64] y a la que aludíamos en el capítulo 3. De esta forma todo patrón de diseño contiene una serie de elementos necesarios que es necesario concretar antes de proceder a su uso en cualquier implementación. Así un patrón de diseño bien documentado facilita extraordinariamente su propio manejo.

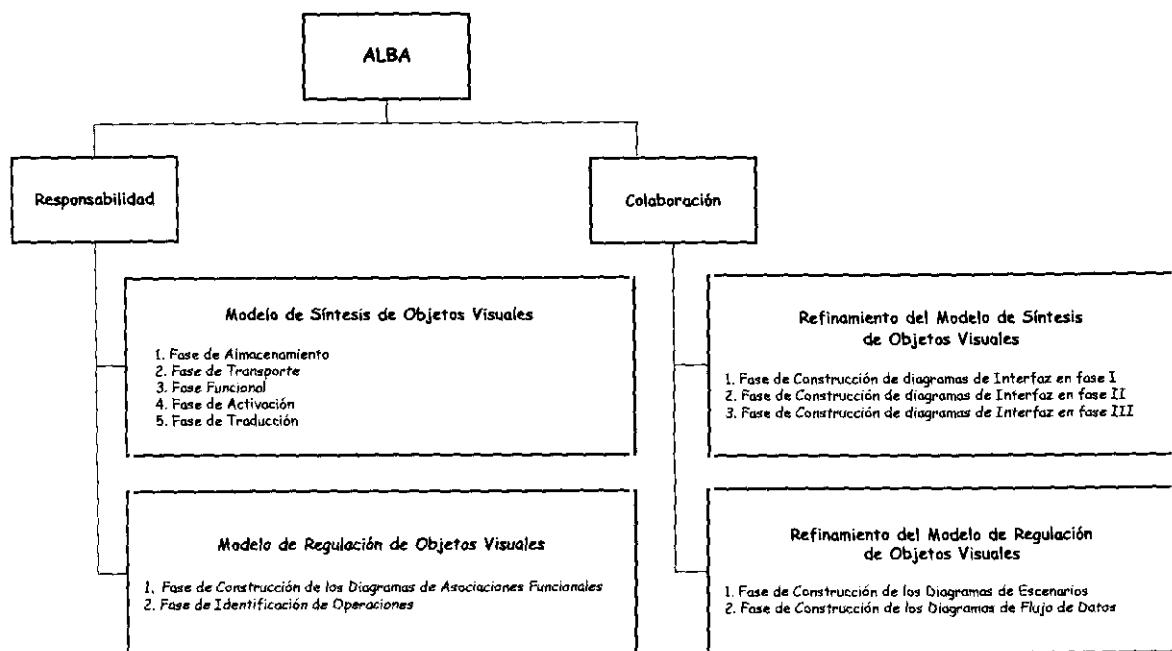


Figura 4.6. Etapas de ALBA

- La etapa de la responsabilidad consiste en delimitar cada una de las clases existentes y las funciones que tiene cada una por separado. Si nos fijamos en el patrón evolutivo que nos sirve como molde, nos encontramos que, en un principio, se crean el ARNm y ARNt. A continuación se dilucida cuál va a ser la funcionalidad que el organismo precisa y, con

arreglo a ello, determinados aminoácidos son transportados hasta el ARNm donde se irán uniendo unos con otros para formar las proteínas, que responderán a la funcionalidad que necesite dicho organismo. Pues bien, en ALBA ocurre exactamente lo mismo. Durante esta etapa definimos, en un principio, unas clases mensajeras y unas clases gestoras, equivalentes al ARNm y ARNt, respectivamente. Las clases mensajeras almacenan la información y las clases gestoras tienen como responsabilidad intervenir en el paso de la información desde las tablas o unidades de almacenamiento hasta las clases mensajeras.

Uno de los puntos débiles de las metodologías orientadas a objetos es el salto existente entre las tablas y los objetos, es decir, entre el modelo de datos y el modelo de objetos [172, 24]. Así las clases gestoras sirven de puente entre ambos tipos de información.

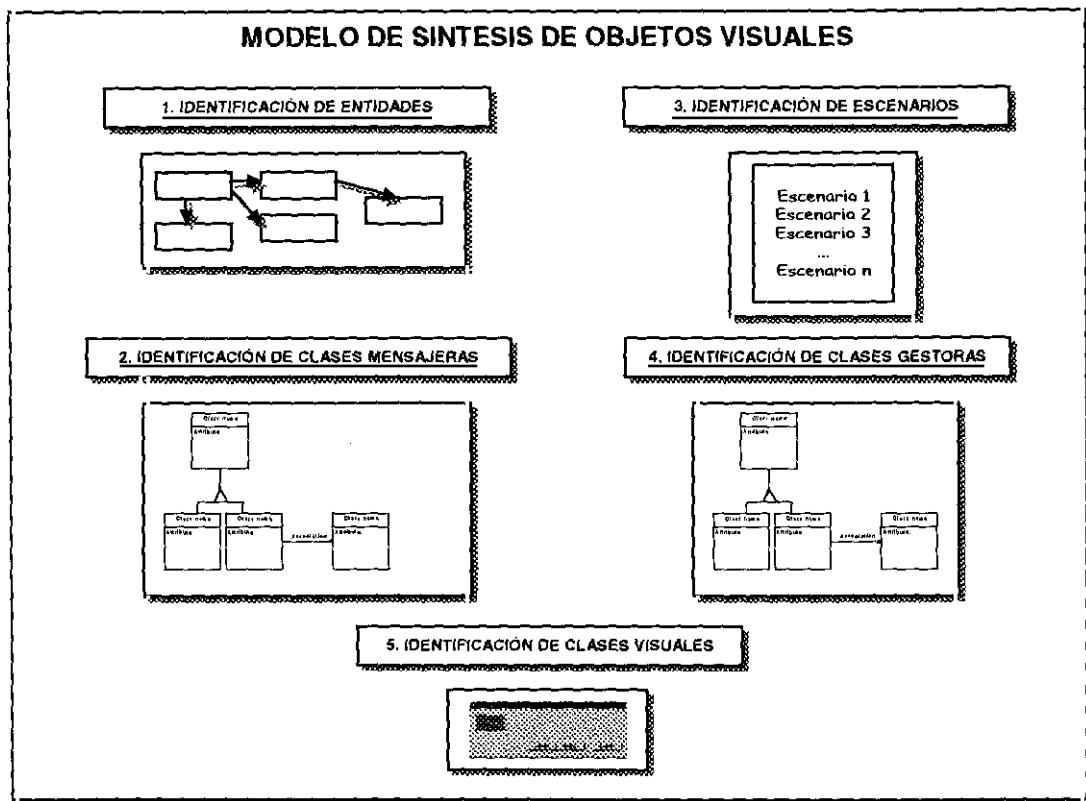


Figura 4.7. Notación del modelo de síntesis de objetos visuales

- b) La colaboración consiste en cómo las clases anteriormente creadas van a colaborar entre sí para llevar a cabo una funcionalidad concreta. La colaboración se plasma en el primer patrón como la interacción de las clases entre sí, y en el segundo patrón, como el paso de mensajes entre las diferentes clases para culminar una determinada funcionalidad.

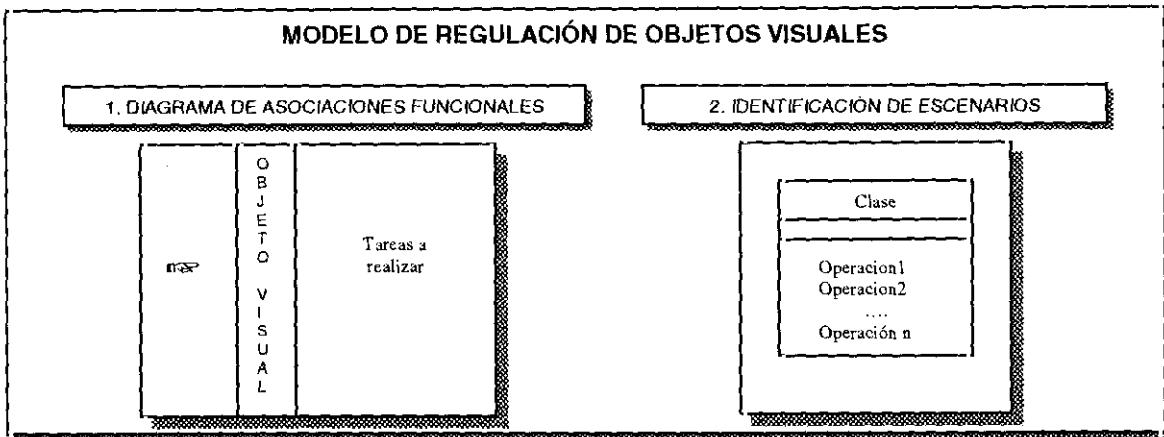


Figura 4.8. Notación del modelo de regulación de objetos visuales

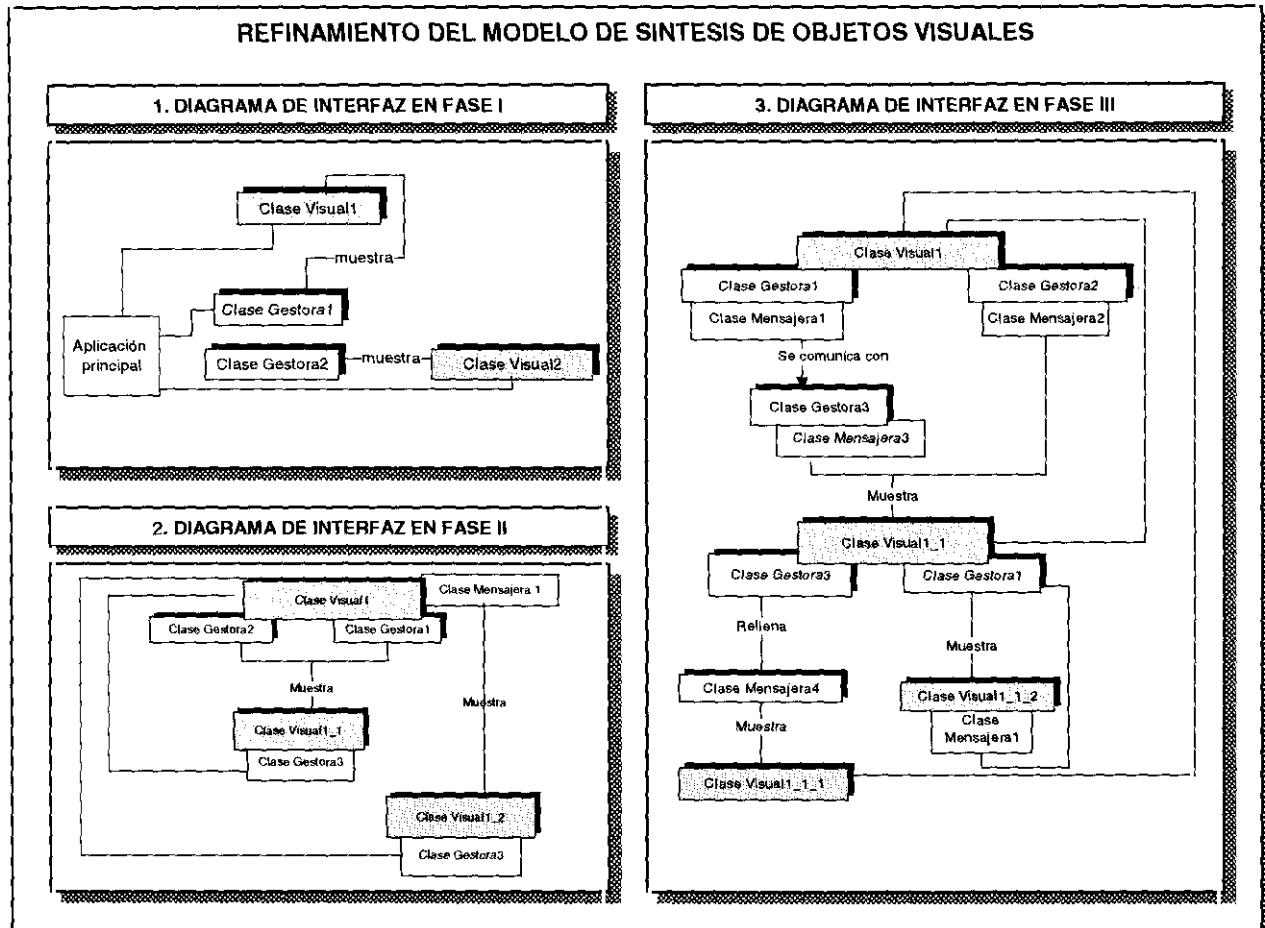


Figura 4.9. Notación del refinamiento del modelo de síntesis de objetos visuales

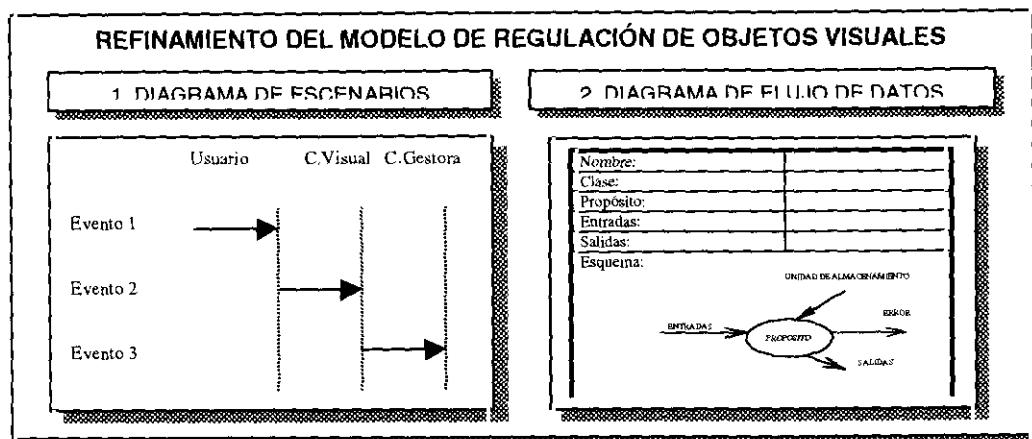


Figura 4.10. Notación del refinamiento del modelo de regulación de objetos visuales

#### 4.2.6. Participantes y Colaboraciones

El lenguaje de patrones propuesto pone de manifiesto el desarrollo en tres capas que cualquier sistema informático debe poseer, tal y como se ilustra en la figura 4.11 y que detallamos a continuación.

- Lógica de datos.** Capa referente al almacenamiento; coincide con las dos primeras fases del modelo de síntesis de objetos visuales, en que se elaboran las unidades de almacenamiento y las clases mensajeras.
- Lógica de la aplicación.** Capa referente al funcionamiento del sistema, coincide con las dos siguientes fases del mismo modelo, en que se elaboran las clases gestoras.
- Interfaz gráfica de usuario.** Capa referente a la interacción directa con el usuario, coincide con la última fase de dicho modelo, en que se elaboran las clases visuales.

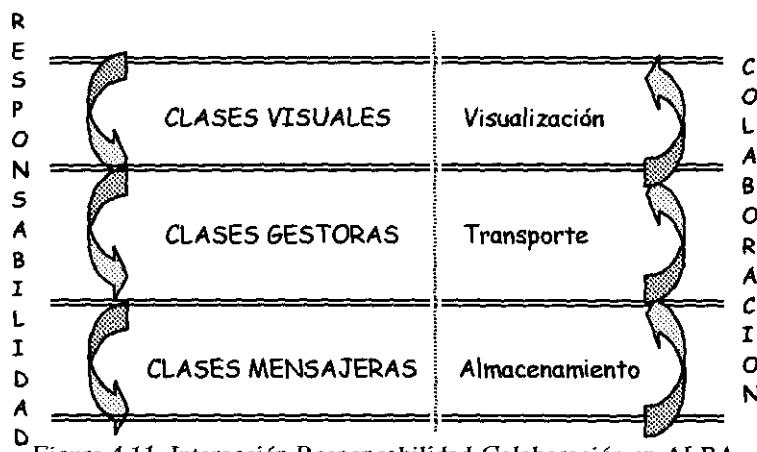


Figura 4.11. Interacción Responsabilidad-Colaboración en ALBA

ALBA presenta como participantes las clases mensajeras, las clases gestoras y las clases visuales, cada una de ellas con una responsabilidad concreta según la capa a la que pertenezca. Las clases mensajeras tienen la responsabilidad de almacenar la información, las clases gestoras recogen o guardan dicha información, y las clases visuales la muestran al usuario. Existe pues un continuo trasiego de datos protagonizado por las clases gestoras que han de colaborar con las clases de almacenamiento para extraer, guardar o actualizar la información, y con las clases visuales para proporcionar las pantallas que muestren dicha información. En la figura 4.11. podemos observar un esquema del modelo de tres capas, de forma que la responsabilidad recae sobre las clases y la colaboración sobre la función específica de cada una.

## **Capítulo 5**

---

### **Lenguaje de patrones para la Síntesis y Regulación de Objetos Visuales**

#### **5.1. Modelo de Síntesis de Objetos Visuales**

##### **5.1.1. Nombre y Objetivo**

El patrón de diseño propuesto se denomina ‘Modelo o Patrón de Síntesis de Objetos Visuales’, abreviadamente PASOV, y su nombre es debido a que su objetivo consiste en ir dilucidando, a través de fases inspiradas en el patrón evolutivo de la biosíntesis de proteínas, las pantallas o clases visuales del sistema informático a diseñar. Estas clases crean los Objetos Visuales, y tanto por cómo se elaboran, como por sus características estructurales y de tipo, guardan una concordancia con las proteínas. Por medio de PASOV y las fases y subfases que incorpora, tal y como se detalla en la figura 5.1. se pueden concretar todas las clases que intervienen en el sistema y que proporcionan una función

concreta. Podemos afirmar que PASOV, junto con el siguiente patrón de diseño que describiremos más adelante, culminan la responsabilidad conceptual y funcional que todo desarrollo requiere.

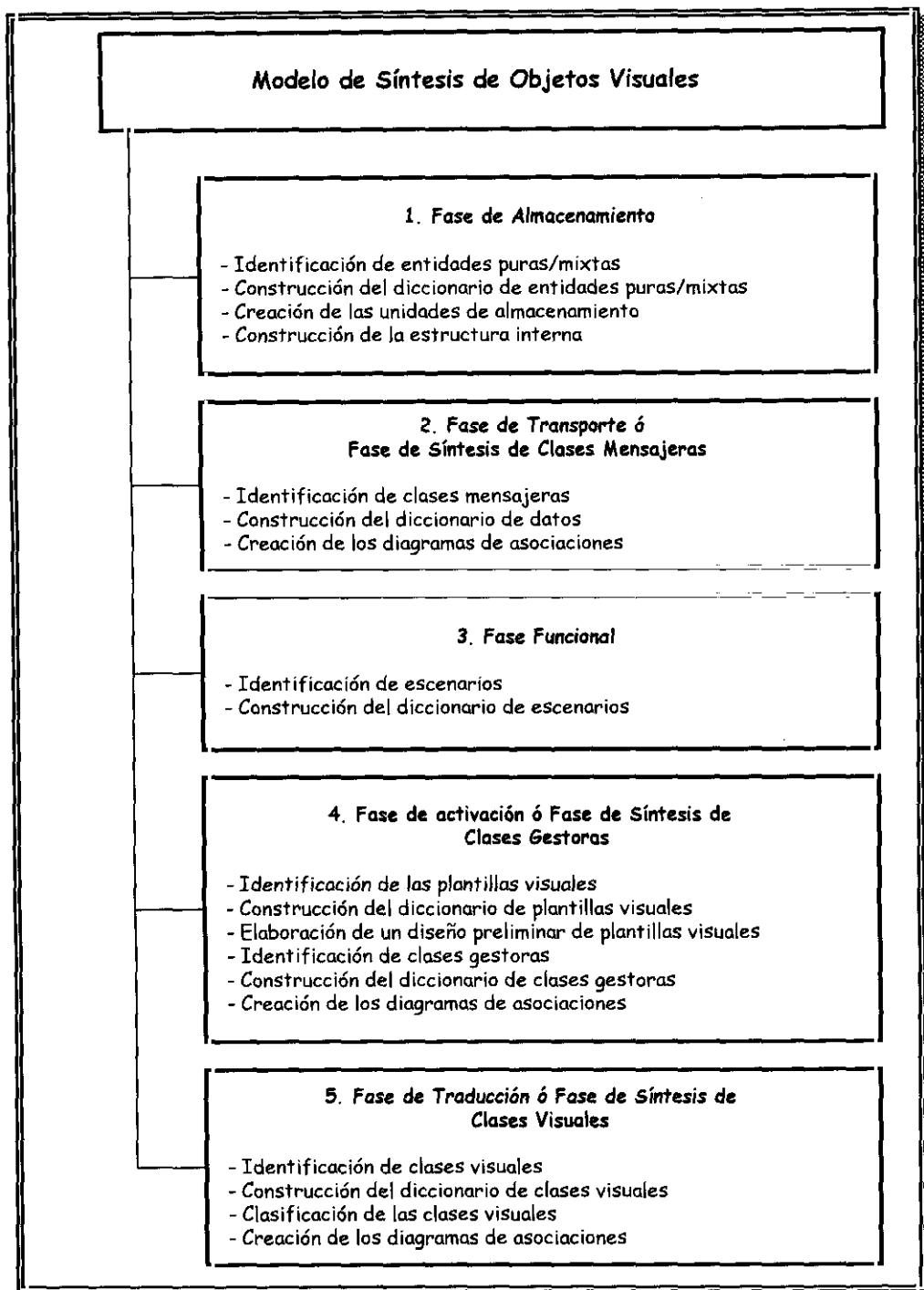


Figura 5.1. Fases y subfases del modelo de síntesis de objetos visuales

Este patrón presenta una **jurisdicción de clase**, ya que es el dominio a que se aplica y una **caracterización de creación**, puesto que el objetivo de este patrón es la construcción de los diferentes objetos que intervienen en determinadas tareas, siguiendo las pautas de clasificación de patrones propuestas por Gamma [64].

### 5.1.2. Estructura

El modelo de Síntesis de Objetos Visuales proporciona la descripción del sistema desde un punto de vista conceptual, de forma que, partiendo de los estratos más internos – donde se encuentra almacenada la información – se va avanzando a través de fases sucesivas, hasta llegar a aquellos objetos – los objetos visuales – que van a interaccionar de manera directa con el usuario. Todo este proceso requiere seguir de una forma iterativa un conjunto de subfases que se van repitiendo a lo largo del ciclo, extendiendo cada vez más las asociaciones y los atributos de cada una de las entidades. El resultado de este modelo es la elaboración de dos tipos de documentación: los **Diccionarios de Clases** y los **Diagramas de Asociaciones**, que más adelante pasaremos a describir.

El modelo de Síntesis de Objetos Visuales se inspira en el proceso de la biosíntesis de proteínas, cuyo esquema central se basa en la secuencia de Crick, que establece la relación entre la ordenación lineal de bases en los ácidos nucleicos con la de los aminoácidos en las proteínas. El eje conductor de PASOV es el que se ilustra en la figura 5.2, y consta de las fases siguientes:

- a) Fase de Almacenamiento
- b) Fase de Transporte
- c) Fase Funcional
- d) Fase de Activación
- e) Fase de Traducción

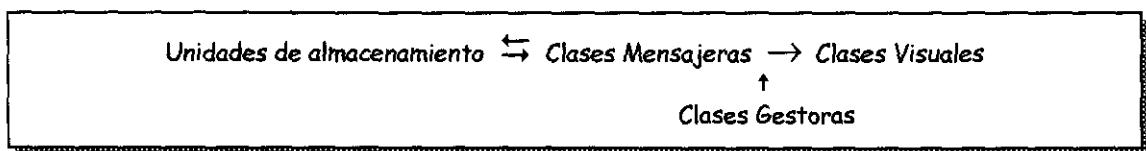


Figura 5.2. Dogma central del modelo de síntesis de objetos visuales

Cada una de estas fases, está constituida por un conjunto de subfases, cuya descripción se detalla a continuación y cuya aplicación se muestra en el anexo que se acompaña, mediante la elaboración de un caso práctico. La figura 5.3. muestra gráficamente el dogma de la figura 5.2, plasmando lo que ocurre desde el almacenamiento inicial de la información, hasta que finalmente ésta es mostrada al usuario. Las unidades de almacenamiento son una forma de guardar la información de manera física, después ésta pasa a un almacenamiento lógico, es decir, a memoria, y de memoria pasa a la aplicación, ya tratable por el usuario, a través de unos transportadores denominados ‘clases gestoras’, constituyendo lo que se llama la interfaz gráfica del usuario.

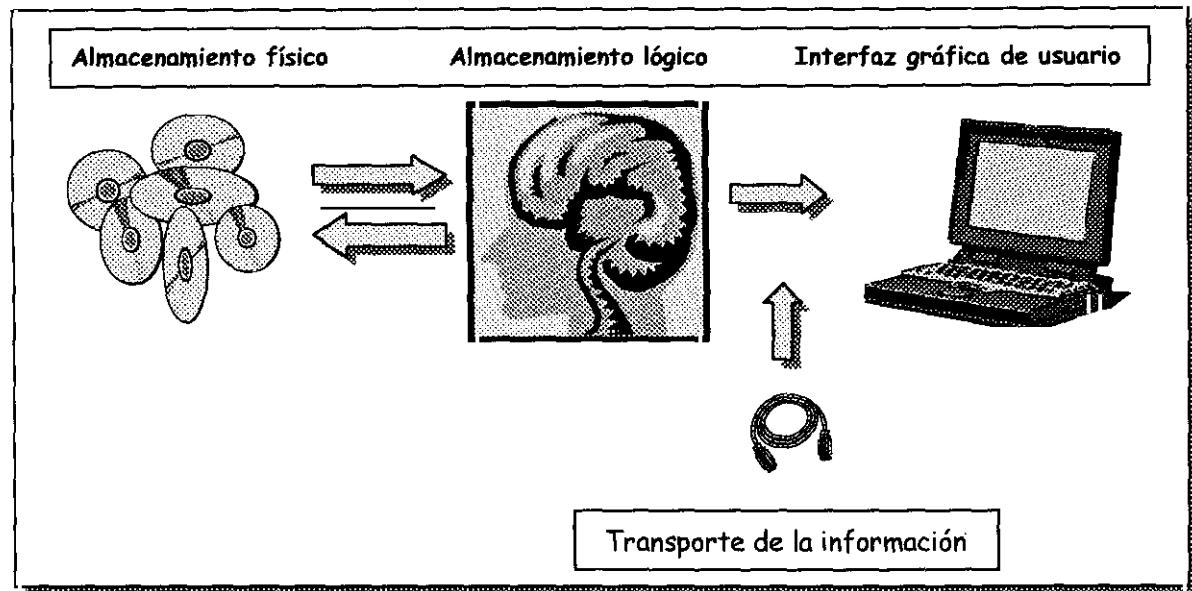


Figura 5.3. Dogma gráfico del modelo de síntesis de objetos visuales

El modelo de síntesis de objetos visuales consta de una serie de fases que en esencia van repitiendo los mismos pasos a lo largo de su desarrollo, como ocurre con la construcción de diccionarios o los diagramas de asociaciones y que explicamos a continuación:

- La construcción de los **diccionarios** obedece a un patrón común, independiente del tipo de diccionarios que sea. Consisten siempre en una lista de las entidades, clases o escenarios a los que haga referencia, seguido de una breve descripción del mismo junto con sus elementos miembro y atributos más característicos. La notación de los diccionarios se especifica en el apéndice A del anexo y es propia del autor.
- La construcción de los **diagramas de asociaciones** es equivalente a los diagramas de objetos de O.M.T. [149]. En el apéndice A del anexo se expone los diferentes tipos de asociaciones que pueden aparecer en los mismos.

### 5.1.2.1. Fase de Almacenamiento

Una primera pregunta que se plantearon los estudiosos de los mecanismos evolutivos de la naturaleza fue la siguiente:

#### ¿Dónde reside la información genética?

Durante la elaboración de un desarrollo informático, dicha pregunta también es el punto de comienzo y normalmente la respuesta suele ser o bien en una estructura de ficheros o bien en una base de datos.

Lo primero a conocer de un sistema es el conjunto de entidades que va a manejar, es decir, los componentes más simples de que va a constar la información a introducir. Normalmente la extracción de dichas entidades simples no suele ser suficiente sino que, por relación de unas con otras, aparecen unas entidades mixtas que constituyen realmente una extensión de las anteriores. Viene a ser equivalente a los componentes más simples por los que está formado el ADN: carbono, hidrógeno, nitrógeno, que por la unión entre ellos mediante enlaces da lugar a la formación de las bases púricas y pirimidínicas.

Una vez que se dispone de las entidades necesarias para el sistema, se agrupan en forma de ficheros y se busca la forma de relacionar unos con otros. Así, se dispone de una información perfectamente empaquetada con una estructura relacional característica.

En la figura 5.4. se muestra la equivalencia entre el almacenamiento a escala biológica y a escala informática, representado en el ADN y en un fichero tradicional, respectivamente.

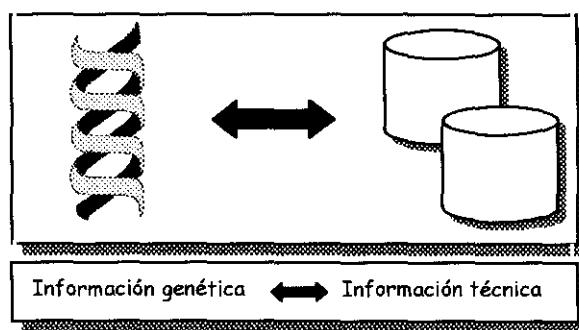


Figura 5.4. Equivalencia de información

Al igual que el ADN muestra un aspecto de doble hélice con las bases púricas y pirimidínicas unidas mediante enlaces covalentes entre sí, en PASOV ocurre algo parecido. Sabemos que en el caso del ADN sólo determinadas bases pueden aparearse con otras, así los enlaces permitidos son: A-T y

G-C; y además, la unión de las bases se realizará tan sólo entre determinadas posiciones de las bases púricas y pirimidínicas. En la base de datos ocurre lo mismo:

- Sólo podrán relacionarse entre sí determinadas tablas y no todas con todas.
- Sólo se podrán unir mediante las claves secundarias y no por cualquier campo.

En la figura 5.5. observamos un estudio comparativo de la fase de almacenamiento en el proceso biológico y en el proceso informático a un nivel más detallado que la figura 5.4, poniéndose de manifiesto cómo se mantiene las uniones entre la información en el ADN, mediante enlaces covalentes, y cómo se crean las relaciones entre las tablas mediante claves ajenas.

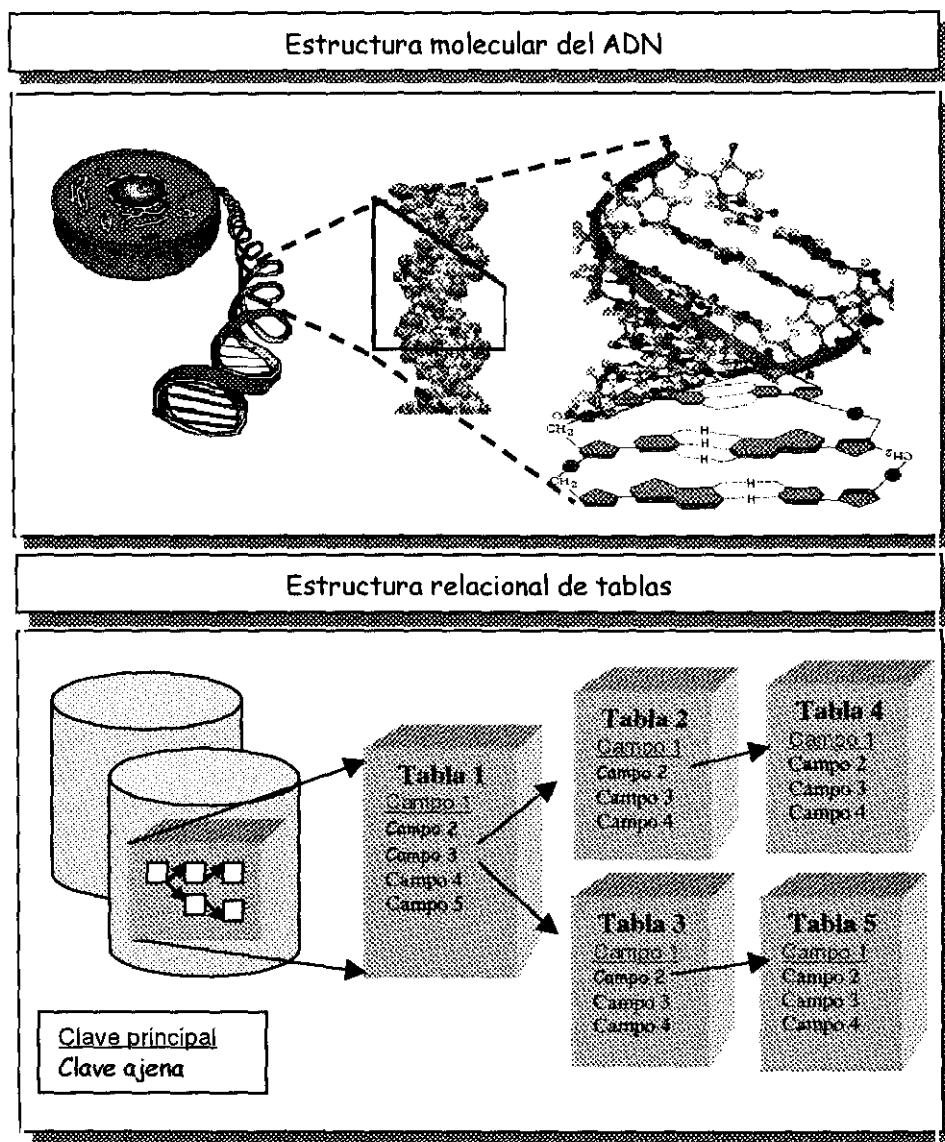


Figura 5.5. Comparación de las fases de almacenamiento biológica e informática

Hemos convenido en subdividir todo el proceso de almacenamiento anteriormente expuesto en seis subfases que se exponen a continuación, describiendo brevemente cada una de ellas.

- a) Identificación de entidades puras.
  - b) Identificación de entidades mixtas.
  - c) Construcción del Diccionario de entidades puras.
  - d) Construcción del Diccionario de entidades mixtas.
  - e) Creación de las unidades de almacenamiento.
  - f) Construcción de la estructura interna.
- 
- a) Identificación de entidades puras..- Las entidades simples o puras son los conceptos que va a manejar una aplicación, sin descender a detalles. Mediante esta subfase concretaremos el objetivo de la aplicación, y extraeremos los conceptos generales que van a ser tratados en la misma.
  - b) Identificación de entidades mixtas..- Las entidades mixtas son una extensión a las entidades anteriormente extraídas. Se denominan mixtas porque agrupan dos o más entidades puras. Durante esta subfase vamos a extraer las entidades mixtas existentes.
  - c) Construcción del diccionario de entidades puras.- Esta subfase consiste en la creación de un diccionario en el que se enumeran cada una de las entidades simples junto con una breve descripción y sus elementos miembro más característicos.
  - d) Construcción del diccionario de entidades mixtas.- Mediante esta subfase se elabora un diccionario en el que se enumeran cada una de las entidades mixtas junto con una breve descripción y sus elementos miembro más característicos.
  - e) Creación de las unidades de almacenamiento.- Durante esta subfase agruparemos las entidades anteriormente descritas en unidades de almacenamiento, también denominadas tablas si descendemos ya al lenguaje de base de datos. En principio esta subfase la hemos basado en los siguientes principios:
    - Cada entidad simple y cada entidad mixta se corresponden con una unidad de almacenamiento
    - Las propiedades de las entidades equivalen a los campos de las tablas.
    - Cada entidad simple se convierte en lo que se denomina una tabla maestra que es aquélla que no precisa relacionarse con ninguna otra para tener integridad por sí misma.

- Cada entidad mixta se convierte en lo que se denomina una tabla asociada. Una tabla asociada es la que se encuentra relacionada con otra, generalmente maestra, y que precisa de ella para tener integridad.
- f) Construcción de la estructura interna.- Cada una de las tablas descritas en la subfase anterior, no son tablas aisladas, sino que están relacionadas entre sí mediante una estructura relacional. En principio esta subfase conlleva las siguientes tareas:
- Todas las tablas han de disponer de una clave única que permita el acceso a sus datos de manera exclusiva y favorezca la unicidad de los mismos.
  - Todas las tablas asociadas han de disponer de una clave secundaria, que permita su relación con la tabla de la que depende.

Establecemos la siguiente conclusión de esta fase, comparando con el proceso de biosíntesis de proteínas:

CUANTO MÁS EVOLUCIONADO SEA UN ORGANISMO, MAS CANTIDAD DE ADN TIENE ⇔ CUANTO MÁS COMPLEJA SEA UNA APLICACIÓN MÁS COMPLEJA ES SU ESTRUCTURA DE ALMACENAMIENTO Y MAYOR VOLUMEN DE DATOS POSEE.

La representación de la estructura interna se hace mediante unos diagramas de cajas y aristas, donde las cajas representan las entidades y las aristas responden a las asociaciones existentes entre ellas. Las asociaciones se representan mediante flechas que van desde la entidad mixta a la entidad pura, especificando la propiedad por la que se relacionan ambas. Este tipo de diagramas no se encuentran en O.M.T., ya que directamente se pasa a las clases sin entrar en la estructura relacional de cada tabla. Sin embargo, es conveniente establecer una definición de las entidades físicas que van a aparecer en la aplicación con el fin de que la transposición a clases sea más o menos inmediata. En el apéndice A del anexo se especifica la notación gráfica de este tipo de relación.

### 5.1.2.2. Fase de Transporte ó Fase de Síntesis de Clases Mensajeras

La siguiente pregunta a plantearnos, una vez que se conoce la residencia de la información genética: el núcleo, es la siguiente:

**¿Quién transporta la información desde el núcleo hasta el citoplasma, lugar donde se traducirá a las proteínas?**

Para que un sistema pueda trabajar con la información de las tablas, necesita una estructura que almacene momentáneamente dicha información, y, por tanto, la transporte desde su almacenamiento físico, en disco duro, hasta su almacenamiento lógico, en memoria.

En la síntesis de proteínas el encargado de esta tarea es el ARNm. En el modelo propuesto surge un nuevo tipo de clases denominadas clases mensajeras, que son clases transportadoras que transfieren la información que han copiado de los ficheros, tomándolo como molde. De ahí que esta fase se conozca también como Fase de Síntesis de Clases Mensajeras. En el momento en que se realiza el copiado de la información de los ficheros a estas clases, éstas pasan a llamarse objetos mensajeros, por lo cual, lo que viaja a memoria son realmente los objetos mensajeros. En la figura 5.6. se establece un estudio comparativo de la fase de transporte en el proceso biológico y en el proceso informático. El proceso biológico tiene, por tanto, como repositorio físico de la información genética, el ADN, que se encuentra localizado en el núcleo celular; este proceso viene a ser equivalente al informático, considerando que las unidades físicas de almacenamiento de la información son las tablas, que residen en disco, siendo éste el equivalente al núcleo celular.

A partir de ahora, todas las fases van a estar constituidas por una serie de subfases más o menos paralelas. En todas ellas se llevará a cabo la identificación de un conjunto de clases particulares, se describirán mediante los diccionarios de clases, se asociarán entre sí, elaborando los diagramas de asociaciones, si cabe, y se identificarán los atributos de que constan. A continuación enumeramos las subfases de esta fase de transporte y describimos brevemente cada una de ellas.

- a) Identificación de las clases mensajeras
  - b) Construcción del diccionario de clases mensajeras
  - c) Creación de los diagramas de asociaciones
- 
- a) Identificación de las clases mensajeras.- En esta subfase se identifica un conjunto de clases construidas a partir de la información facilitada por las unidades de almacenamiento. Son clases que actúan a modo de plantillas, cuya estructura es idéntica a la que se encontraba en las tablas, y que realiza un copiado de la información, que se encuentra en las mismas, para transportarla de disco a memoria, lugar donde se realizarán subsiguientes transformaciones.

- b) Construcción del diccionario de clases mensajeras. - La descripción de las clases mensajeras, junto con sus atributos característicos y las asociaciones que pueda tener con otras clases, se formaliza mediante un diccionario de clases.
- c) Creación de los diagramas de asociaciones. - Las clases mensajeras derivan de clases superiores y se encuentran relacionadas entre sí, creándose diagramas de asociaciones, las cuales, tal y como se expone en O.M.T. [149]. Dichas asociaciones pueden ser de dos tipos: herencia y agregación. En esta subfase, pues, se trata de extraer posibles relaciones de herencia y agregación, entre las clases mensajeras; en los diagramas de asociaciones, también aparecen los atributos, extraídos en la subfase anterior.

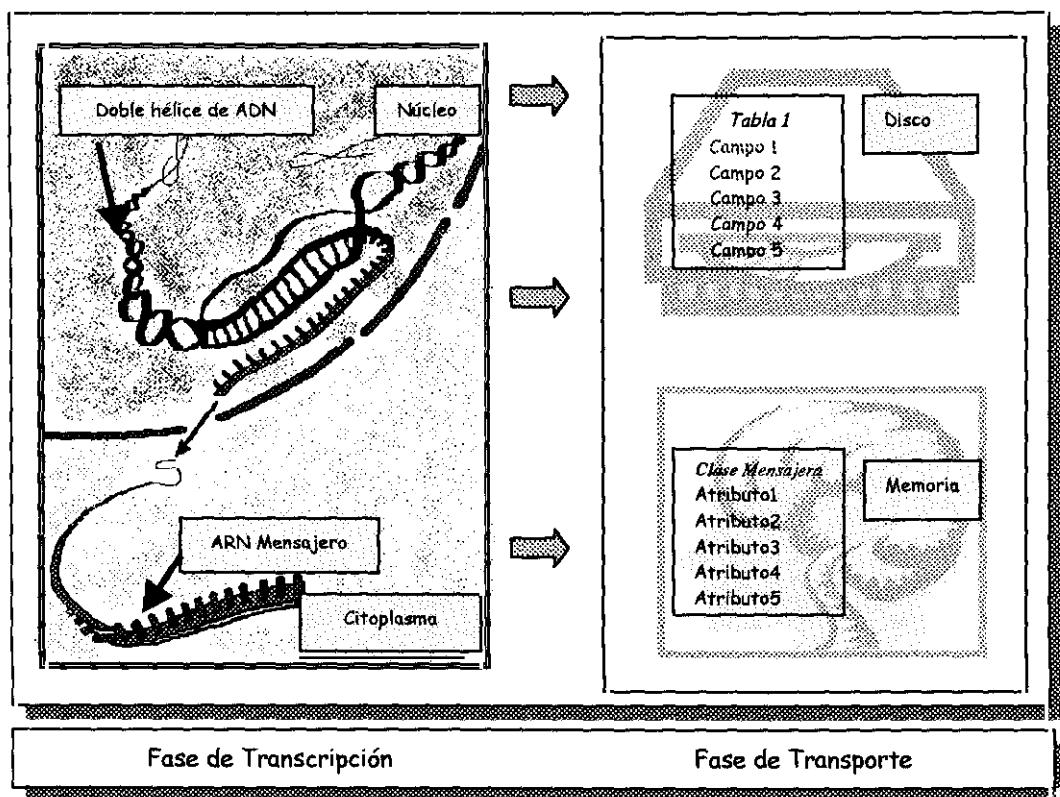


Figura 5.6. Comparación de las fases de transporte biológica e informática

Igualmente se puede establecer una conclusión de la fase:

ES NECESARIA LA PRESENCIA DE INTERMEDIARIOS QUE TRANSCRIBAN FIELMENTE EL MENSAJE GENÉTICO PARA LLEVARLO AL CITOPLASMA. ↔ ES NECESARIA LA PRESENCIA DE CLASES INTERMEDIAS QUE TRANSPORTEN LA INFORMACIÓN DE DISCO A MEMORIA.

### 5.1.2.3. Fase Funcional

En esta fase daremos respuesta a la siguiente pregunta:

#### **¿Qué funciones va a desempeñar la información almacenada?**

Durante la fase funcional es necesario especificar los objetivos del sistema informático, ya que con arreglo a cubrir unas u otras necesidades, se construirán unos u otros objetos visuales. Ocurre lo mismo que en el modelo biológico, en el que existe una serie de funciones a cubrir en el organismo, razón por la cual será necesario sintetizar unas determinadas proteínas y no otras, para llevarlas a cabo. En un sistema informático, especificaremos primeramente las grandes funciones para que, una vez enumeradas, se puedan definir mediante una breves descripciones, dando lugar a los diagramas de escenarios.

En la figura 5.7. mostramos una comparación entre la funcionalidad necesaria a escala biológica y la necesaria en el caso de PASOV. En el proceso biológico podemos observar cómo a partir del ADN del núcleo, por la transcripción, se transmite la información genética al ARNm, y cómo éste interviene en la elaboración de las proteínas, que representan diferentes funcionalidades en la célula. Por otro lado el proceso informático también cubre una serie de aspectos funcionales tal y como se indica en la figura 5.7, como ‘imprimir’, ‘buscar’ o ‘abrir’. Esta funcionalidad se lleva a cabo mediante las plantillas visuales, pero de momento en esta fase aún no han sido elaboradas. La figura muestra las pantallas para entender que la funcionalidad extraída en esta fase, se llevará a cabo mediante las clases visuales que se deducirán en fases posteriores.

La fase funcional se divide en las siguientes subsfases, que enunciamos y describimos a continuación:

- a) Identificación de escenarios
- b) Construcción del diccionario de escenarios
  
- a) Identificación de escenarios.- Mediante ella enumeramos todos los escenarios de la aplicación. Se trata de elaborar una lista donde se vean claras las funciones desencadenadas desde el sistema informático a elaborar.
- b) Construcción del diccionario de escenarios.- Con esta subfase describiremos cada función, anteriormente enunciada, mediante la creación de un diccionario.

De esta fase podemos concluir:

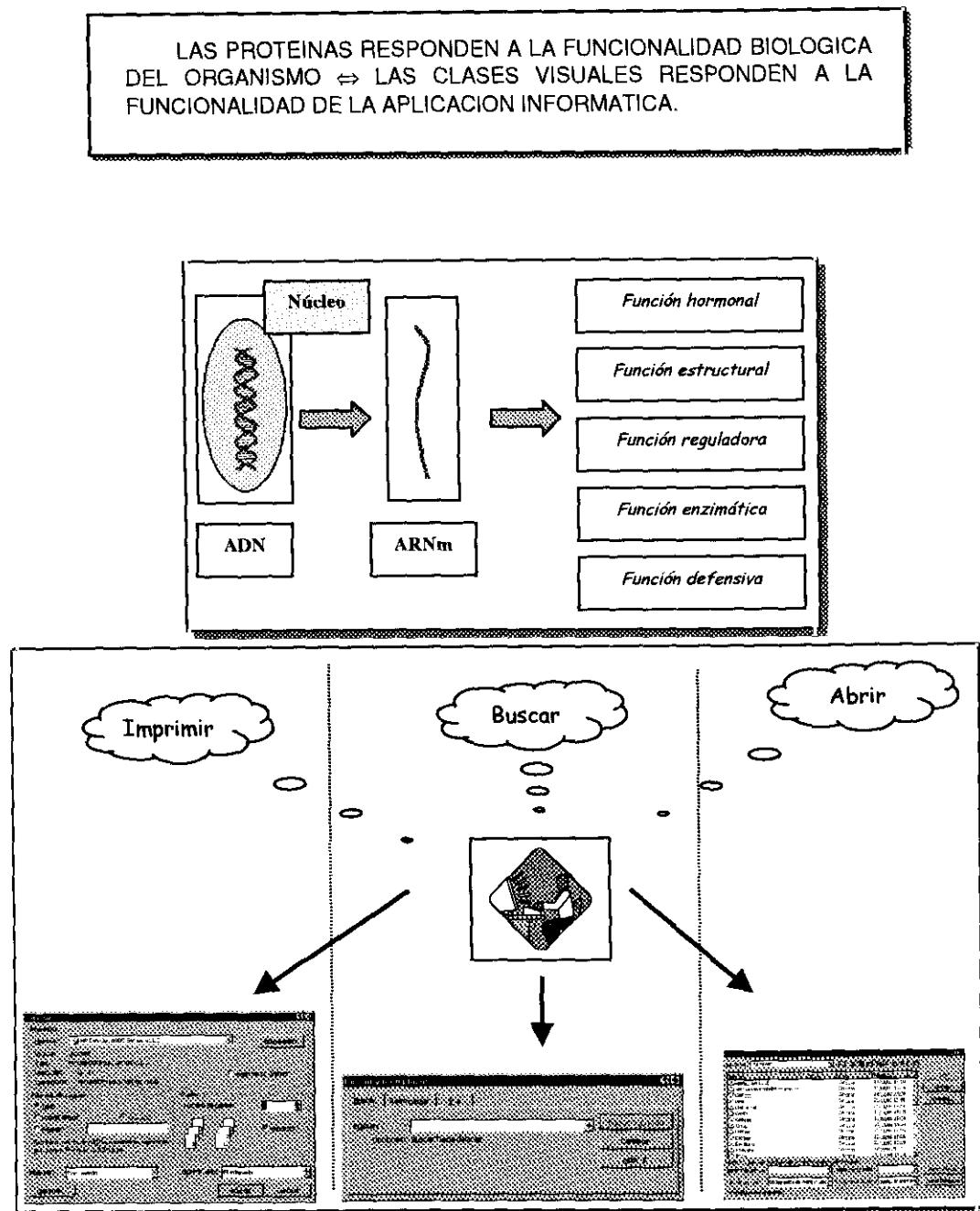


Figura 5.7. Comparación de las fases funcionales biológica e informática

#### 5.1.2.4. Fase de Activación o Fase de Síntesis de Clases Gestoras

Durante esta fase se trata de dar respuesta al interrogante:

**¿Quién recoge las unidades aminoacídicas?**

Las proteínas están formadas por la unión de aminoácidos mediante enlaces polipeptídicos. Los aminoácidos se encuentran dispersos en el citoplasma, por lo que es necesario un elemento que los vaya recogiendo y acercando al lugar donde se llevará a cabo la síntesis de proteínas propiamente dicha; estos elementos, transportadores de los aminoácidos, son moléculas de ARNt. El modelo de síntesis de Objetos Visuales, consta en esta fase de la construcción de las plantillas de los objetos visuales, presentes en memoria y, que precisan igualmente de un transportador que las acerque hasta el lugar de su propia traducción.

En la figura 5.8. se pone de manifiesto una comparación de ambas fases de activación. Por un lado ilustramos la activación de los aminoácidos al unirse al ARNt y por otro la activación de las clases visuales, que en principio son meras plantillas, que, al unirse con la clase mensajera correspondiente – portadora de la información – y a la clase gestora – que ayuda a hacer el trasvase de información de la clase mensajera a la clase visual – aparece como resultado un objeto visual, ahora sí con la información transportada.

Los transportadores de los que hablamos son las clases gestoras y tienen un doble cometido:

- a) Transportar las plantillas hasta el lugar donde se realizará la síntesis de objetos visuales, y,
- b) Actuar como adaptador de las plantillas a la información proporcionada por las clases mensajeras.

Durante esta fase se llevará a cabo la creación de las plantillas visuales, su descripción mediante el diccionario de clases y la especificación de sus atributos; en este caso, ya que se trata de plantillas, podríamos decir atributos visuales puesto que serán recursos del tipo ‘botón’, ‘controles de edición’, ‘controles estáticos’, y otros.

Las plantillas visuales son realmente recursos que se emplean en cualquier herramienta de programación para dar lugar a la interfaz de usuario. Durante esta fase enumeraremos todas las plantillas que precise la aplicación. Normalmente, hay una serie de plantillas visuales que podrían haberse identificado al principio del modelo, puesto que son evidentes, se intuyen en cuanto se sabe qué entidades van a constituir el sistema; sin embargo, hay otras plantillas que no aparecen a primera

vista, sino que son deducidas al enunciar los escenarios de la aplicación. De esta forma, queda justificada la obligada situación de esta fase en PASOV: justo después de haber deducido los escenarios.

Otro tipo de clases a especificar en esta fase: las clases gestoras, que también serán identificadas y descritas mediante los diccionarios. De ella también deduciremos las distintas asociaciones de que consten y concretaremos sus atributos.

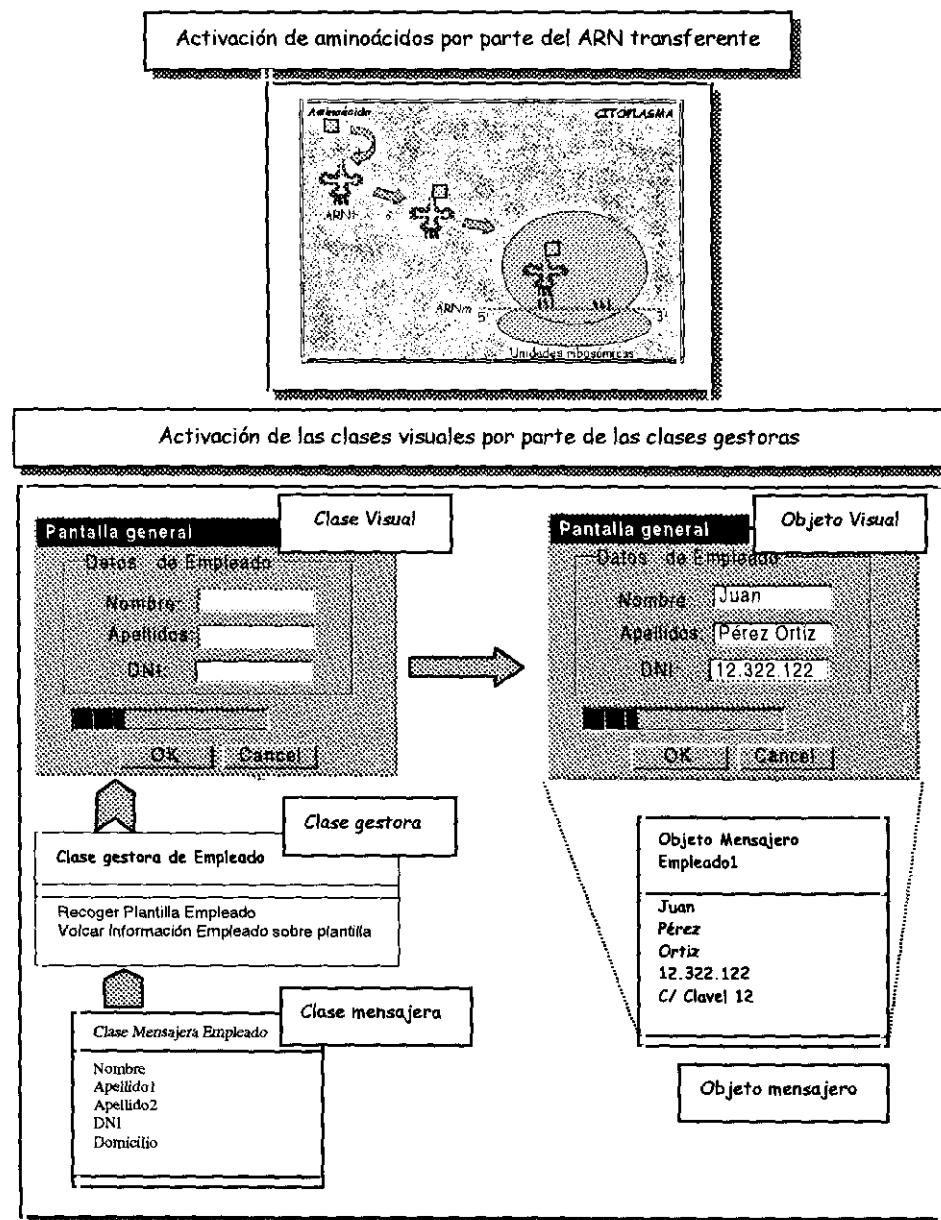


Figura 5.8. Comparación de las fases de activación biológica e informática

A continuación exponemos las subfases de esta fase de activación, haciendo una breve descripción de cada una de ellas:

- a) Identificación de las plantillas visuales.
  - b) Construcción del diccionario de plantillas visuales
  - c) Elaboración de un diseño preliminar de plantillas visuales
  - d) Identificación de las clases gestoras.
  - e) Construcción del diccionario de las clases gestoras
  - f) Creación de los diagramas de asociaciones
- 
- a) Identificación de las plantillas visuales..- Durante esta subfase elaboraremos el diseño de las plantillas visuales basándonos en todas las estructuras de datos extraídas en las fases anteriores, y que resulta necesario mostrar en la aplicación.
  - b) Construcción del diccionario de plantillas visuales..- En este momento describiremos las plantillas visuales anteriormente enunciadas, es decir, qué tipo de información está preparada para que se muestre a través de dichas plantillas.
  - c) Elaboración preliminar de un diseño de plantillas visuales..- En esta subfase procedemos a mostrar un diseño, que no tiene por qué ser el definitivo, pero que puede servir para ver cómo distribuir la información que van a mostrar las plantillas anteriormente enunciadas. Mediante este diseño se procede a:
    - Identificar todos los controles visuales necesarios para mostrar la información descrita en la fase de almacenamiento.
    - Identificar todos los controles visuales necesarios para llevar a cabo la funcionalidad descrita en la fase funcional.

- d) Identificación de las clases gestoras.- En esta subfase identificaremos las clases gestoras, cuya aparición tiene dos objetivos: por un lado el transporte de las plantillas visuales y por otro la supervisión en la fase de traducción.
- e) Construcción del diccionario de las clases gestoras.- En este instante se procede a la descripción de las clases gestoras, y al enunciado de las propiedades más relevantes. También en el diccionario se van a poner de manifiesto las relaciones que estas clases tienen con otras clases.
- f) Construcción del diagrama de asociaciones.- En esta etapa se representan, mediante un diagrama de asociaciones, las relaciones de las clases gestoras con otras clases del sistema.

Como consecuencia de todo lo expuesto se llega a la siguiente conclusión:

ES NECESARIA LA ACTIVACION DE LOS AMINOACIDOS EN EL CITOPLASMA MEDIANTE LA ESTERIFICACION A SUS CORRESPONDIENTES ARN DE TRANSFERENCIA ↔ ES NECESARIA LA ACTIVACION DE LAS PLANTILLAS DE LOS OBJETOS VISUALES MEDIANTE LAS CLASES GESTORAS.

#### 5.1.2.5. Fase de Traducción o Fase de Síntesis de Clases Visuales

Durante esta fase se procede a traducir el lenguaje nucleotídico del código genético al lenguaje aminoacídico, para formar las proteínas. Esta fase constituye la respuesta a la pregunta formulada por Wyatt en 1972:

##### ¿Cuándo la información genética se convierte en conocimiento?

La información guardada en las tablas y transferida de manera temporal a las clases mensajeras, va a ser traducida al lenguaje del usuario final, es decir, a pantallas de datos. Durante esta fase el complejo formado por la clase gestora y la plantilla visual, se une a la clase mensajera para dar lugar a las clases visuales. Estas clases visuales tienen también un tipo y una descripción. Las clases visuales, van a responder a una funcionalidad diferente, según el tipo a que pertenezcan; así, al igual que ocurre con las proteínas, se va a realizar una clasificación de las mismas.

Las subfases de esta fase son las siguientes, pasando a describirlas a continuación:

- a) Identificación de clases visuales

- b) Construcción del diccionario de clases visuales
- c) Clasificación de las clases visuales
- d) Construcción del diagrama de asociaciones

- a) Identificación de clases visuales.- Durante esta subfase identificaremos las clases visuales que constituyen el sistema.
- b) Construcción del diccionario de clases visuales.- En este punto elaboraremos un diccionario en el que se describe cada clase visual, y se establecen las relaciones con otras clases visuales, si las hubiera. En el diccionario también especificaremos los atributos, denominados controles visuales en las clases visuales, que aseguran que se pueda llevar a cabo la funcionalidad.
- c) Clasificación genérica de las clases visuales.- Las clases visuales, según la funcionalidad para la que han sido diseñadas, van a corresponderse con diferentes tipos, de ahí que se realice una clasificación genérica con todas ellas. Esta subfase es ya dependiente del lenguaje de programación visual a utilizar, puesto que detallaremos de qué tipo van a ser las clases visuales a emplear. Entre los tipos que encontramos, son característicos los siguientes:
  - Clase Aplicación.- Se trata de la clase representante de la aplicación. Sólo se permite una de este tipo.
  - Clase Documento.- Es la clase que almacena la información de toda la aplicación. Sólo se permite una de este tipo.
  - Clase Marco.- Es la que soporta recursos globales, tales como el menú principal, la barra de herramientas. Sólo se permite una de este tipo.
  - Clases Contenedor de Carpetas.- Se refiere a clases que contienen múltiples clases carpetas.
  - Clases Carpeta.- Son las que proporcionan información acerca del sistema en general, acerca de la configuración, o permiten la elección de algo, ofreciendo un asistente, para que resulte más sencilla la tarea. Puede haber tantas como se deseé.
  - Clases Vista.- Actúan como visores de información, aunque también permiten el mantenimiento de la misma. Su número es ilimitado.

- Clases Diálogo. - Se trata de las clases que interaccionan con el usuario directamente estableciendo un diálogo con él. Puede haber tantas como sea necesario.
- d) Construcción del diagrama de asociaciones.- Durante esta subfase se construyen los diagramas que establecen las distintas asociaciones, de unas clases con otras. En estos diagramas se ponen de manifiesto dos tipos de asociaciones: herencia y agregación.

En la figura 5.9. ilustramos la comparación entre la fase de traducción del proceso biológico y la fase de traducción del proceso informático.

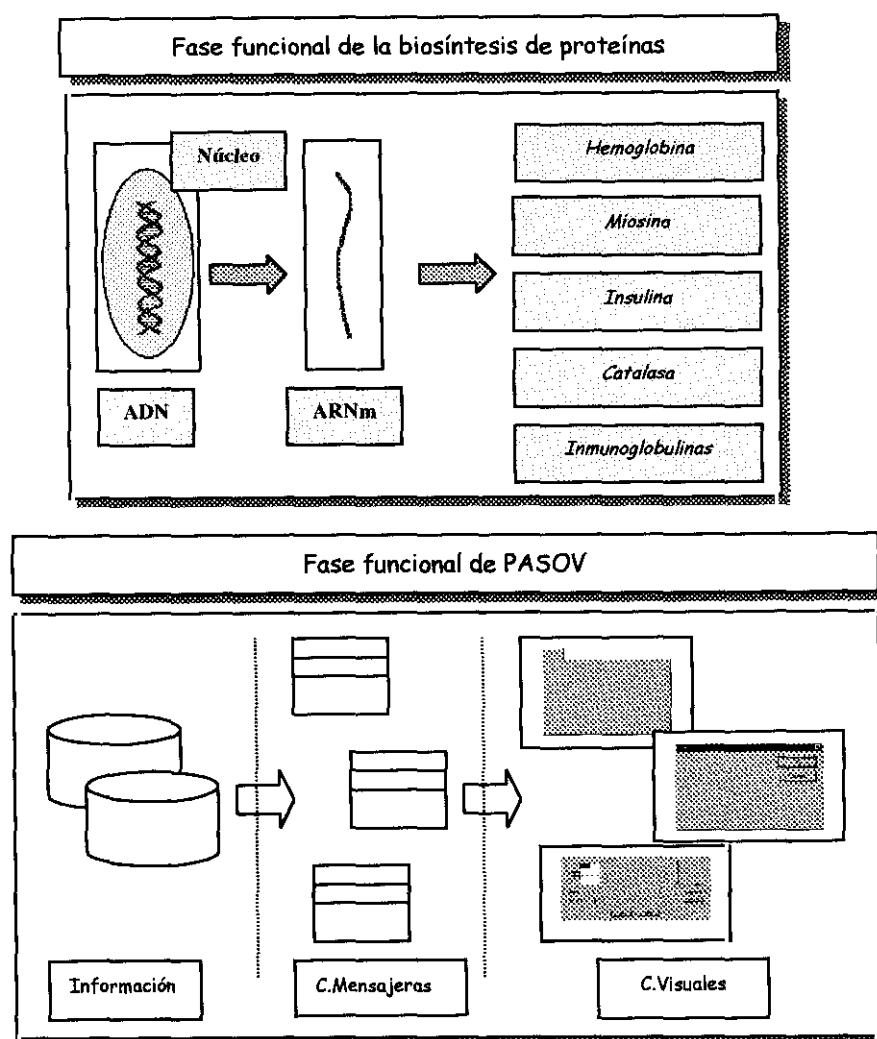


Figura 5.9. Comparación de las fases de traducción biológica e informática

Como ejemplo del proceso biológico mostramos la misma imagen que se empleó en la figura 5.8, para poner de manifiesto las diferentes funciones en que intervienen las proteínas, con la diferencia de que ahora ya hemos plasmado un ejemplo de proteína que posea la funcionalidad descrita. Por otro lado la fase de traducción en el ámbito informático muestra un aspecto general de diferentes clases visuales, según la funcionalidad que desempeñen en la aplicación a desarrollar. En este caso se ilustra una clase visual tipo ‘carpeta’, ‘vista’ y ‘diálogo’.

La conclusión de esta fase resulta ser la siguiente:

ES NECESARIO TRADUCIR EL LENGUAJE DE LOS NUCLEOTIDOS DEL CODIGO GENETICO AL DE LOS AMINOACIDOS DE LAS PROTEINAS ⇔ ES NECESARIO TRADUCIR LA INFORMACION DE LOS FICHEROS A INFORMACION VISIBLE Y MANEJABLE POR EL USUARIO, ES DECIR, A LAS CLASES VISUALES

### 5.1.3. Participantes y colaboraciones

Los participantes en este patrón de diseño son, como ya hemos venido anunciando durante el desarrollo del mismo, las clases mensajeras, las clases gestoras y las clases visuales. Las tres deben colaborar entre sí para llevar a cabo las funciones para las que han sido creadas; puesto que se ha descrito este aspecto con amplitud, no consideramos oportuno extendernos más en este apartado.

### 5.1.4. Aplicación

El patrón de diseño PASOV se puede aplicar a diferentes problemas susceptibles de ser implementados en una herramienta informática. Nosotros detallaremos una aplicación concreta basándonos en el caso práctico que nos ocupa. Presentaremos un patrón de diseño específico para el módulo de representación del conocimiento de un generador de S.B.C. cuando se admiten tres tipos de unidades del conocimiento: reglas, marcos y marco causales.

## PATRON PARA REPRESENTAR EL CONOCIMIENTO

### Objetivo

Proporcionar la interfaz para un generador de S.B.C. que maneje tres tipos de representación del conocimiento: Reglas, marcos y marcos causales.

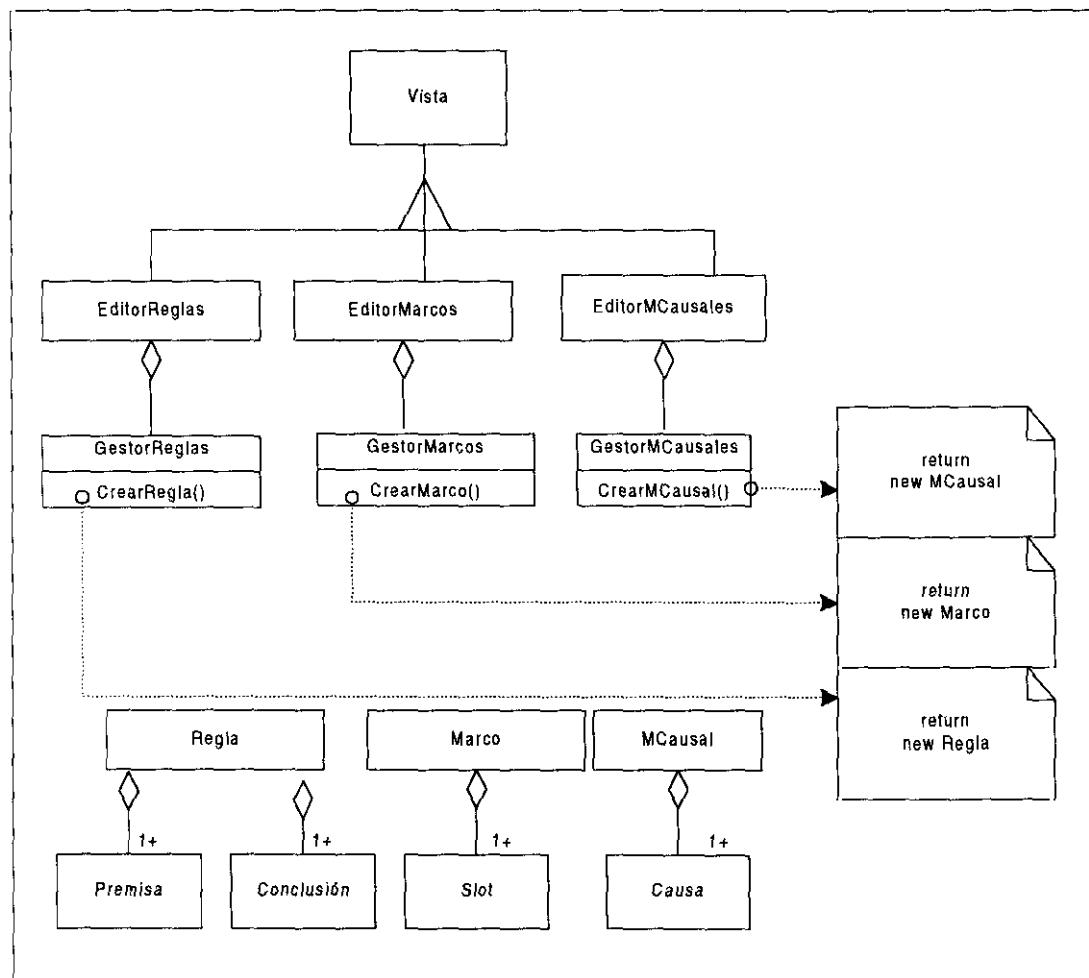
### Participantes

- **EditorReglas**.- Interfaz para la introducción de reglas de producción. Es una clase visual tipo 'vista'.
- **EditorMarcos**.- Interfaz para la introducción de marcos. Es una clase visual tipo 'vista'.
- **EditorMCausales**.- Interfaz para la introducción de marcos causales. Es una clase visual tipo 'vista'.
- **GestorReglas**, **GestorMarcos**, **GestorMCausales**.- Clases que manejan toda la información relativa a reglas, marcos y marcos causales. Son clases intermedias entre la unidad de conocimiento propiamente dicha, es decir, entre la regla, marco ó marco causal y la pantalla que lo visualiza: EditorReglas, EditorMarcos y EditorMCausales.

### Colaboraciones

Los tres editores habrán de colaborar con sus clases gestoras correspondientes, con el fin de que éstas gestionen las unidades de conocimiento que se introducen a través del editor. A través de las clases gestoras por tanto, se crean las distintas unidades de conocimiento. Por otro lado cada clase que representa a cada unidad de conocimiento, colabora con otras clases por relaciones de agregación, que determinan sus clases componentes.

### Diagrama



## 5.2. Modelo de regulación de Objetos Visuales

### 5.2.1. Nombre y Objetivo

Al igual que el objetivo de las proteínas en el organismo es cumplir la función o funciones para las que han sido creadas. El patrón o modelo de regulación de objetos visuales, también denominado PAROV, cubre el objetivo de detallar la funcionalidad concreta de cada una de las clases que se han creado en el patrón anterior. Con este patrón se cubre la responsabilidad global del sistema.

Como sabemos, en la naturaleza las proteínas no actúan por si mismas, sino que se ayudan de factores liberadores o energéticos. Los objetos visuales también necesitan de la ayuda de unos factores para completar su funcionalidad. Estos factores son las clases gestoras, que ya han aparecido en una etapa anterior.

Puesto que estas funciones se asocian a objetos visuales, se establece la relación de ambos, de ahí el nombre para el patrón. Pero cada función se desglosa en una serie de operaciones, que vuelven a identificarse, y a asociarse de nuevo a los objetos visuales que los lleven a cabo. Por tanto las fases de que consta este patrón, según se muestra en la figura 5.10 son: la construcción de los diagramas de asociaciones funcionales y la identificación de operaciones.

Este patrón presenta una **jurisdicción de clase**, ya que es el dominio a que se aplica y una **caracterización de comportamiento**, puesto que en este patrón lo que se trata es de extraer la responsabilidad concreta de cada clase.

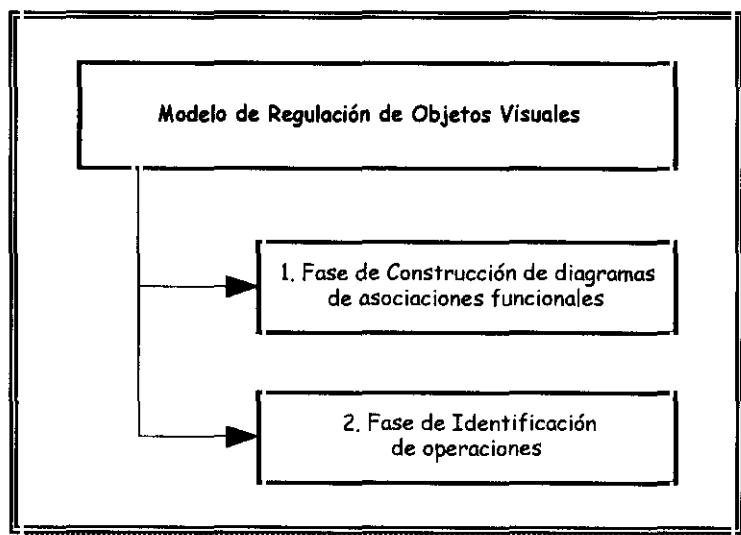


Figura 5.10. Fases del modelo de regulación de objetos visuales

## 5.2.2. Estructura

La estructura interna del modelo de regulación de objetos visuales consta de las dos fases anteriormente identificadas. Ambas se documentan mediante dos tipos de recursos que se explican a continuación:

- a) La construcción de los diagramas de asociaciones funcionales se representa mediante unos diagramas, introducidos por el autor, que consisten en exponer tareas que son necesarias para que los escenarios, descritos en la fase funcional, puedan llevarse a cabo. Estos diagramas consisten en un cuadro, en cuyo interior se van describiendo mediante una frase, cada operación a realizar, separándolas de la acción del usuario mediante una barrera. Con dicha barrera queremos poner de manifiesto el objeto visual que tiene delante el usuario, a través del cual, se desencadenan dichas tareas. La notación de estos diagramas se especifica en el apéndice A del anexo y es propia del autor.
- b) La identificación de operaciones se realiza empleando la representación de las clases, en el espacio dedicado a la especificación de las funciones miembro, tal y como se detalla en el apéndice A del anexo.

### 5.2.2.1. Fase de Construcción de diagramas de asociaciones funcionales

Durante esta fase se procede a establecer qué tareas van a ser desencadenadas desde los objetos visuales extraídos en el patrón anterior, como consecuencia de cualquier acción que pueda emprender el usuario.

### 5.2.2.2. Fase de Identificación de Operaciones

Las tareas extraídas en la fase anterior se formalizan dándoles un nombre, especificando quién va a desencadenar cada una de ellas. La forma de representar las operaciones es asociándolas a la clase a la cual pertenecen tal y como se muestra en el apéndice A del anexo.

## 5.2.3. Participantes y colaboraciones

Los participantes protagonistas de este patrón son los objetos visuales y las colaboraciones son las que se establecen con el resto de clases, mensajeras y gestoras, para llevar a cabo la funcionalidad que se desencadena por los eventos de usuario a través de los objetos visuales. Se cubre de esta forma la responsabilidad funcional de todo sistema informático, que, junto con el patrón anterior que cubría

la responsabilidad conceptual, llegamos a justificar el porqué hemos denominado ‘responsabilidad’ al bloque formado por estos dos primeros patrones de diseño genéricos de ALBA.

#### 5.2.4. Aplicación

El patrón de diseño PAROV se puede aplicar específicamente a aquellos tipos de problemas que precisen procesos de cálculo por el desencadenamiento de un evento concreto del usuario, a través de un control visual. Hemos detallado como ejemplo, un patrón específico extraído del caso práctico del generador de S.B.C.: la propagación de la incertidumbre siguiendo los tres mecanismos elegidos: probabilidad subjetiva, factores de certeza y redes bayesianas.

Con este patrón ejemplo se trata de plasmar cómo a partir de la petición de un usuario solicitando la propagación de la incertidumbre, son desencadenadas de forma secuencial, una serie de funciones encargadas de dicha propagación.

Finalmente, como conclusión del modelo de regulación de objetos visuales, establecemos la siguiente:

LAS PROTEINAS SON FUNDAMENTALES EN TODOS LOS ASPECTOS DE LA ESTRUCTURA Y FUNCION CELULARES ⇔ LOS OBJETOS VISUALES SON LOS RESIDENTES DE LA ESTRUCTURA Y FUNCIONALIDAD DE LA INFORMACION QUE VAYA A MANEJAR LA APLICACION.

## PATRON PARA PROPAGAR LA INCERTIDUMBRE

### Objetivo

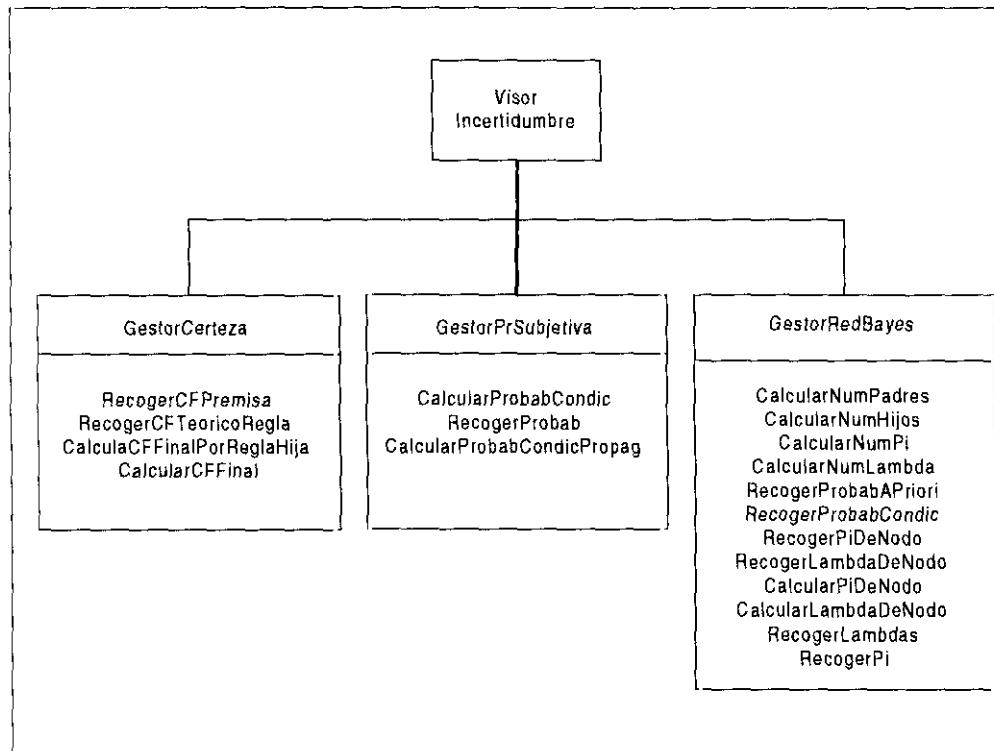
Proporcionar la funcionalidad necesaria para llevar a cabo la propagación según factores de certeza, probabilidad subjetiva o redes bayesianas.

### Participantes

- **VisorIncertidumbre**.- Se trata de la clase visual donde se va poniendo de manifiesto la propagación según el mecanismo elegido. Es una clase visual tipo ‘vista’.
- **GestorCerteza**.- Se trata de la clase gestora encargada de la propagación de la incertidumbre basándose en la teoría de los factores de certeza.
- **GestorPrSubjetiva**.- Se trata de la clase gestora encargada de la propagación de la incertidumbre basándose en la probabilidad subjetiva.
- **GestorRedBayes**.- Se trata de la clase gestora encargada de la propagación de la incertidumbre basándose una red bayesiana.

Cada una de estas clases tiene detalladas las funciones necesarias para llevar a cabo el proceso de propagación, según el mecanismo elegido.

### Diagrama



## 5.3. Refinamiento del Modelo de Síntesis de Objetos Visuales

### 5.3.1. Nombre y Objetivo

El tercer patrón de diseño que proponemos en ALBA está relacionado directamente con la colaboración conceptual, y describe cómo los objetos van a interaccionar unos con otros para responder a la funcionalidad del sistema. Su nombre abreviado es PARSOV, y en él se especifica el encadenamiento de unos objetos visuales con otros, a través de los controles visuales de que constan. Esta etapa viene a ser equivalente a lo que constituye la estructura cuaternaria de las proteínas. Las proteínas, vistas superficialmente, muestran un aspecto diferente que si nos vamos adentrando en su interior donde iremos descubriendo la aparición de una estructura interna compleja constituida por la unión de varias cadenas polipeptídicas. Pues bien, durante la colaboración conceptual ocurre lo mismo, uno se va adentrando en la estructura de los objetos visuales, para observar que algunos están constituidos por un esqueleto interno, resultado de la unión de varios objetos visuales entre sí. En la figura 5.11. se exponen las fases de que consta el refinamiento que nos ocupa.

Este patrón presenta una **jurisdicción de objeto**, ya que es el dominio a que se aplica y una **caracterización de comportamiento**, puesto que en este patrón lo que se trata es de extraer las colaboraciones entre los objetos que intervienen en el sistema.

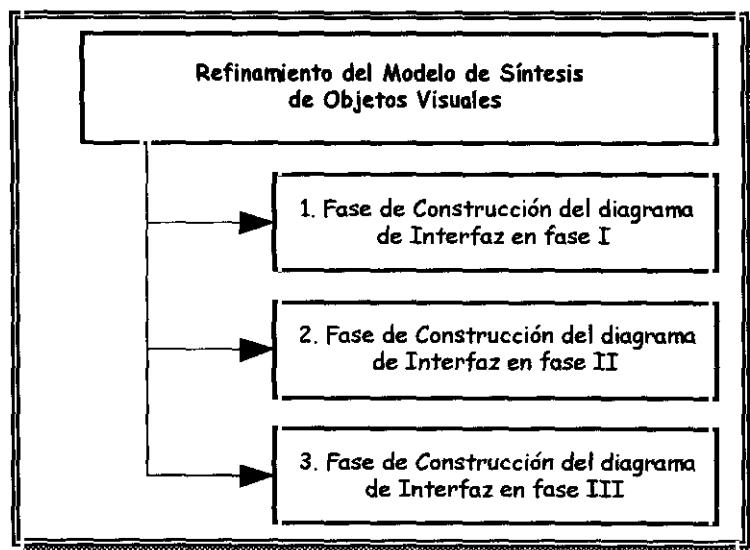


Figura 5.11. Fases del refinamiento del modelo de síntesis de objetos visuales

### 5.3.2. Estructura

En el patrón de diseño PASROV y, puesto que ya conocemos la funcionalidad de los objetos visuales, se refinan éstos mediante la aparición de controles visuales tales como puntos de menú o botones, que desencadenan las operaciones expuestas en fases anteriores. De esta forma dividimos este refinamiento en tres fases, que se corresponden con tres vistas de la aplicación, aumentándose el nivel de complejidad y siendo semánticamente coherentes las tres vistas entre sí; estas fases vienen determinadas por la creación de los diagramas de interfaz, que, junto con los diccionarios de clases extraídos en el primer patrón descrito, constituyen parte de la documentación que consideramos fundamental para elaborar el análisis y diseño.

En la figura 5.12. exponemos vistas sucesivas de la aplicación, como si estuviéramos viendo la aplicación a diferentes aumentos. La explicación viene a ser que en un primer momento lo que el usuario ve de la aplicación es el menú principal: **diagrama de interfaz en fase I**. En un segundo momento se pueden abrir diferentes pantallas pulsando directamente los puntos de menú que tiene a su alcance, en este caso nos encontramos ante el **diagrama de interfaz en fase II**. Si el usuario se centra en cada una de las pantallas y comienza a navegar por las opciones de botón que dichas pantallas le proporciona, estamos ante el **diagrama de interfaz en fase III**.

Cada uno de estos diagramas se corresponde con una fase en PASROV, que a continuación pasamos a detallar.

#### 5.3.2.1. Fase de construcción del diagrama de interfaz en fase I

En esta fase estableceremos las opciones que la aplicación ofrece sin entrar en demasiado detalle; se trata de las opciones generales de la aplicación. Para ello emplearemos los diagramas de interfaz en fase I, que exponen los objetos visuales que aparecen, en un primer nivel de complejidad de la aplicación, como consecuencia de la comunicación entre el sistema informático y el usuario.

#### 5.3.2.2. Fase de construcción de diagramas de interfaz en fase II

En esta fase descendemos a un segundo nivel de complejidad; se trata de representar la funcionalidad de cada uno de los objetos visuales extraídos en la fase anterior. En esta fase juegan un gran papel los controles visuales de los objetos visuales. La representación de los diagramas de interfaz en fase II viene a ser similar a la explicada en el apartado anterior.

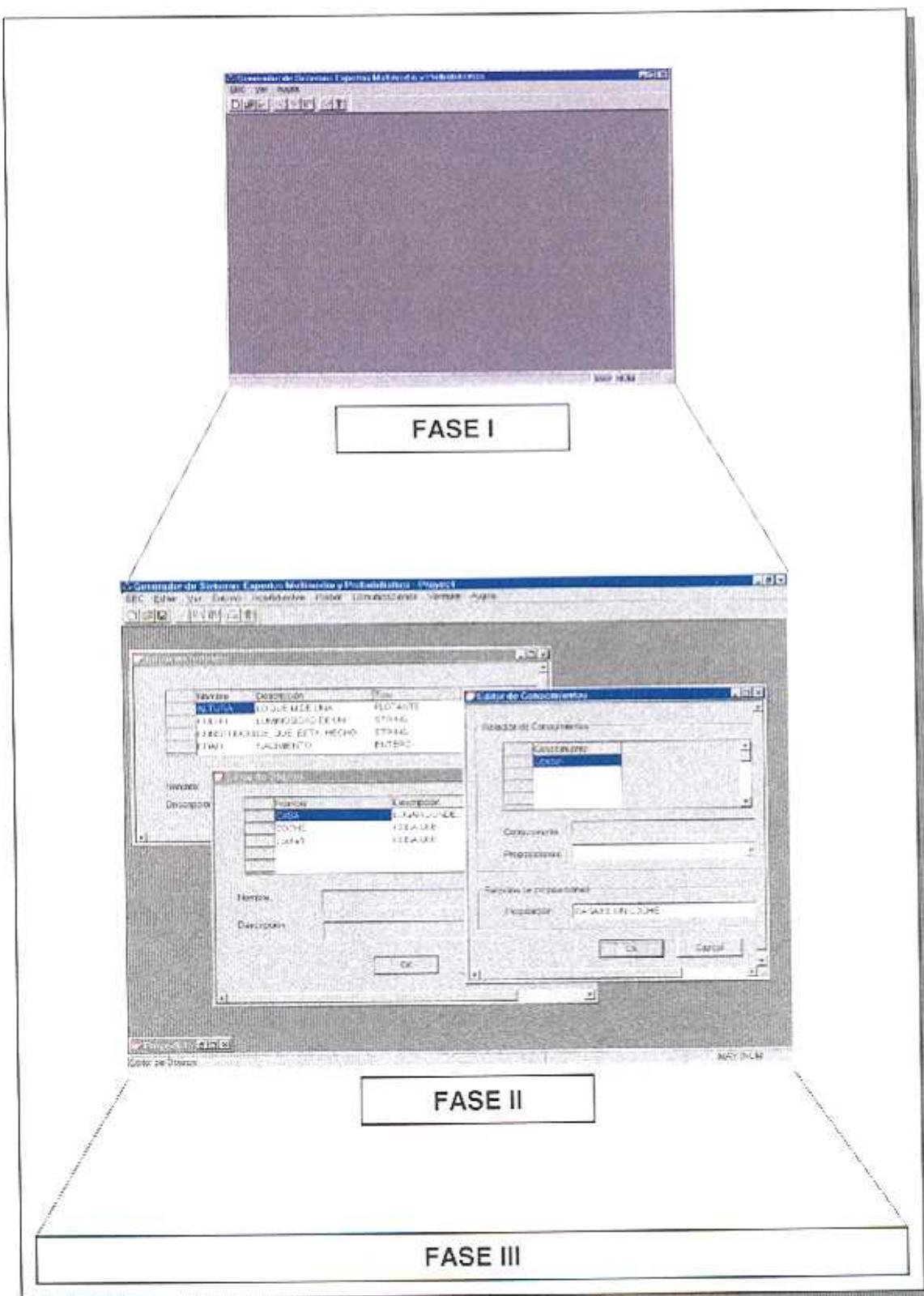


Figura 5.12. Fases I y II del refinamiento del modelo de síntesis de objetos visuales

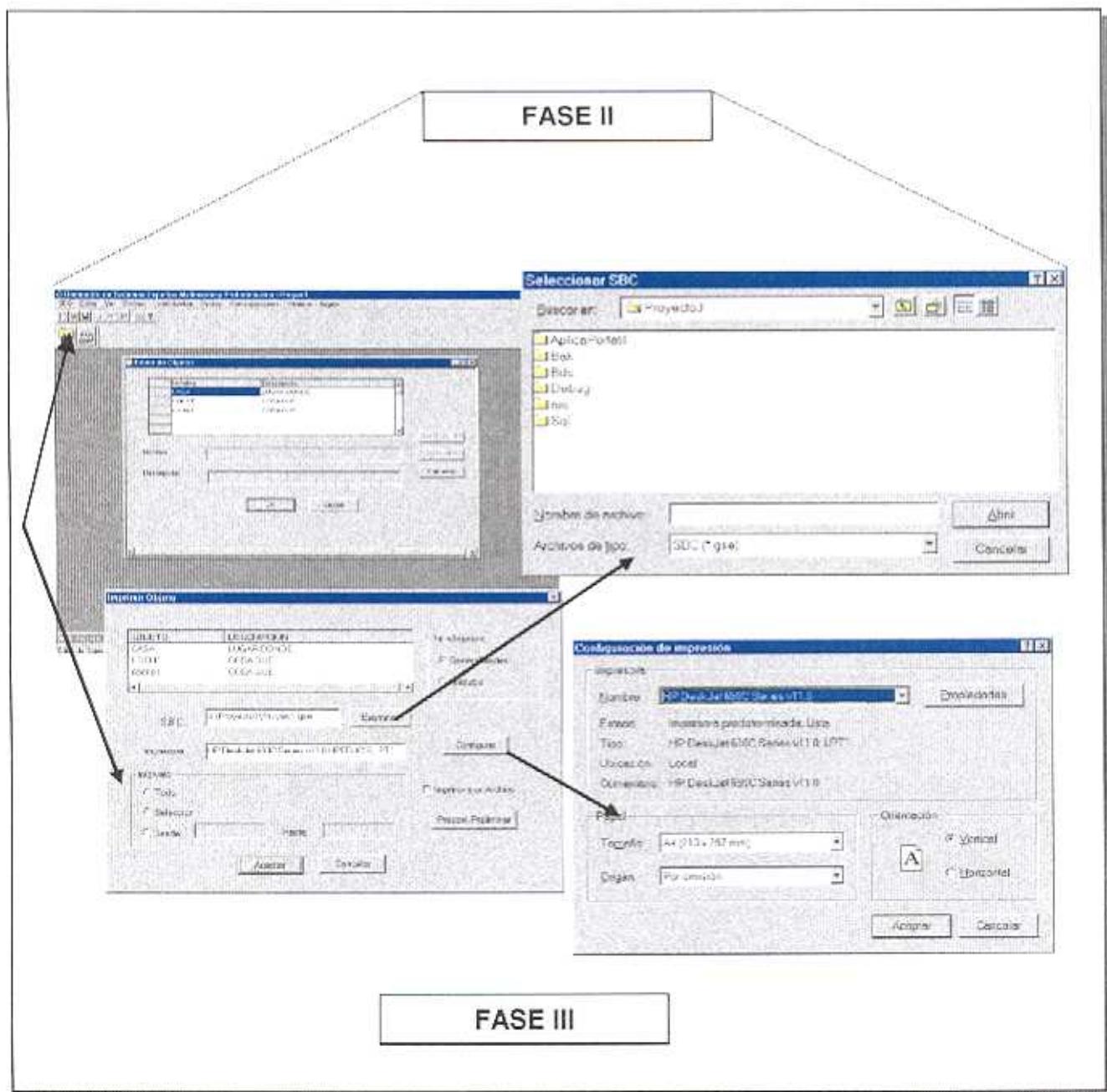


Figura 5.13. Fase III del refinamiento del modelo de síntesis de objetos visuales

### 5.3.2.3. Fase de construcción de diagramas de interfaz en fase III

En esta fase se desciende a un tercer nivel de complejidad y ya se trata de ir viendo cada una de las opciones de cada objeto visual, analizando a su vez qué otros objetos visuales desencadena.

Estos tres tipos de diagramas han sido introducidos por el autor, ya que en O.M.T. no existe ninguno que exprese lo que ocurre al nivel de la interfaz de usuario. Hemos utilizado una simbología especial para los distintos tipos de clases que existen en este diagrama, cuya notación se explica en el apéndice A del anexo, tanto para las clases visuales, gestoras como para las clases mensajeras. Los mensajes desencadenados se expresan a modo de etiqueta de la asociación que une ambas clases.

### 5.3.3. Participantes y colaboraciones

Los participantes de este patrón son los objetos visuales y las colaboraciones se establecen a través de los controles visuales que contienen, para desencadenar otros objetos visuales. Con este patrón se pone de manifiesto la colaboración conceptual, al tener que colaborar unos objetos visuales con otros.

### 5.3.4. Aplicación

La aplicación elegida para mostrar el funcionamiento de este patrón de diseño ha sido la opción 'imprimir', que por las distintas opciones que presenta, parece idóneo para ilustrar un ejemplo de colaboración conceptual. Este caso ejemplo ha sido extraído también del ejemplo desarrollado en el anexo.

La conclusión del refinamiento del modelo de síntesis de objetos visuales resulta ser la siguiente:

LAS PROTEÍNAS DISPONEN DE UNA ESTRUCTURA TRIDIMENSIONAL QUE LES DOTA DE CIERTA ESPECIFICIDAD ⇔ LOS OBJETOS VISUALES TIENEN UNA ESTRUCTURA CONSTITUIDA POR CONTROLES VISUALES QUE LES PROPORCIONA UN CIERTO NIVEL DE NAVEGACIÓN POR LA APLICACIÓN INFORMÁTICA.

## PATRON PARA IMPRIMIR UN ARCHIVO

### Objetivo

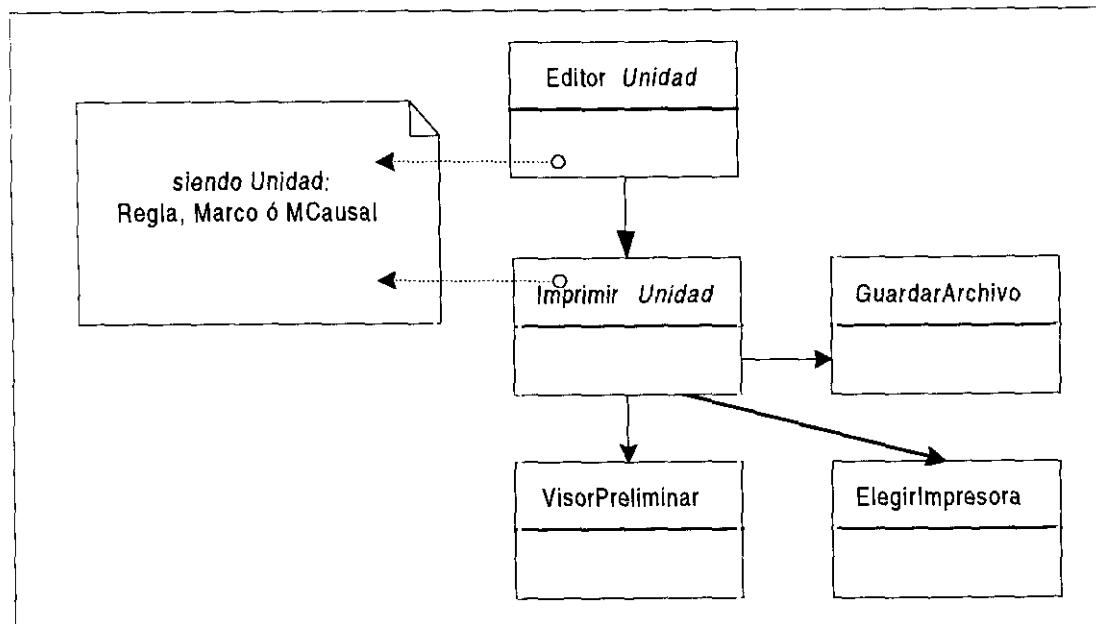
Proporcionar la interfaz necesaria para confeccionar un determinado fichero con información del S.B.C. para después imprimirllo si se desea.

### Participantes

- **EditorUnidad**.- Clase editor de cualquiera de las tres unidades de conocimiento que admite el S.B.C. Podría ser: **EditorReglas**; **EditorMarcos**; **EditorMCausales**.
- **ImprimirUnidad**.- Clase que da a elegir el tipo de información que se desea obtener. Podría ser: **ImprimirRegla**; **ImprimirMarco**; **ImprimirMCausal**. Estas clases son clases visuales tipo: 'diálogo'.
- **GuardarArchivo**.- Clase que ofrece el árbol de directorios para que se elija el nombre del fichero a crear y su trayectoria. Es una clase visual tipo: 'diálogo'.
- **ElegirImpresora**.- Clase que visualiza las opciones de impresión, además de las impresoras instaladas en el equipo. Es una clase visual tipo: 'diálogo'.
- **VisorPreliminar**.- Clase que proporciona una visión del archivo a imprimir. Es una clase visual tipo: 'diálogo'.

Cada una de estas clases colaboran entre sí para llevar a cabo la impresión de un determinado fichero o la visión preliminar del mismo, sin necesidad de imprimir, según lo que se deseé.

### Diagrama



## 5.4. Refinamiento del Modelo de Regulación de Objetos Visuales

### 5.4.1. Nombre y Objetivo

El cuarto patrón de diseño del lenguaje de patrones propuesto consiste en un refinamiento del modelo de regulación de objetos visuales, PARROV, y en él se trata de especificar la colaboración funcional existente entre los diferentes objetos para llevar a cabo la funcionalidad del sistema. Este objetivo se cubre mediante las dos fases especificadas en la figura 5.14.

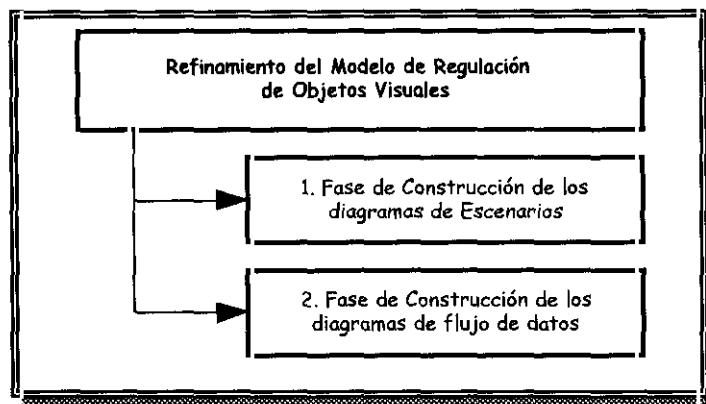


Figura 5.14. Fases del refinamiento del modelo de regulación de objetos visuales

Este patrón presenta una **jurisdicción de clase**, ya que es el dominio a que se aplica y una **caracterización de comportamiento**, puesto que en este patrón lo que se trata es de extraer las colaboraciones funcionales existentes.

### 5.4.2. Estructura

Como sabemos, cada proteína lleva a cabo una función o funciones concretas en el organismo. Cada una de estas funciones engloba una serie de mecanismos particulares agrupados bajo el epígrafe de dicha funcionalidad. Pues bien, este patrón de diseño tiene un desarrollo paralelo. En él hemos detallado en un primer nivel los escenarios, equivalentes a las funciones en las proteínas. A continuación cada escenario lo hemos desglosado en pequeñas tareas que han de desencadenarse secuencialmente para que se cumpla correctamente la funcionalidad asociada a cada escenario. Esto se lleva a cabo mediante la consecución de dos fases, que a continuación pasaremos a describir:

### 5.4.2.1. Fase de construcción de los diagramas de escenarios

Durante esta fase elaboraremos los diagramas de escenarios, consistentes en una representación gráfica de cada una de las funciones extraídas en la fase funcional, especificando los objetos que los desencadenan y las operaciones de que constan. Estos diagramas de escenarios son los que O.M.T. denomina: ‘diagramas de seguimiento de sucesos para un escenario’. Ofrecen de una forma muy clara, un esquema de cómo una acción del usuario, repercute hasta el núcleo de la aplicación, desencadenando una serie de mensajes, que o bien vuelven de nuevo al usuario, o bien desencadenan otros eventos antes de devolver el control de nuevo al usuario.

### 5.4.2.2. Fase de construcción de los diagramas de flujo de datos

Durante esta fase especificamos cada una de las operaciones extraídas en el modelo de regulación de objetos visuales, concretando las unidades de almacenamiento de donde necesitan extraer la información y especificando qué tipo de salidas va a dar dicha operación. Para representar los diagramas de flujo de datos se han empleado los que O.M.T. proporciona.

En la figura 5.15. se expone la regulación de objetos visuales en el proceso informático. El diagrama de escenarios representa de forma esquemática el conjunto de operaciones que se realizan en un momento dado, pero no hay que perder de vista que cada una de estas operaciones, realmente equivale a un conjunto de entradas y salidas y a una serie de procesos, que se representan por los diagramas de flujo de datos, tal y como se representa en la figura 5.15.

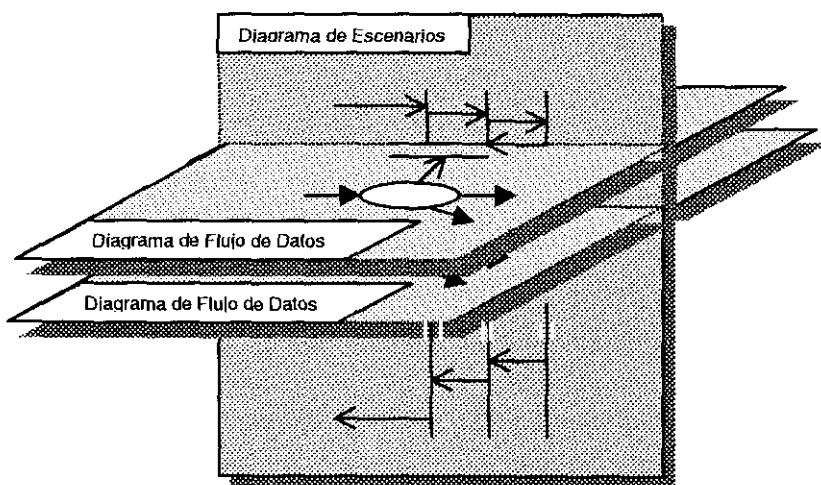


Figura 5.15. Vista gráfica del refinamiento del modelo de regulación de objetos visuales

### 5.4.3. Participantes y colaboraciones

Los participantes de este cuarto patrón son los tres tipos de clases: mensajeras, gestoras y visuales, puesto que este patrón cubre por completo la colaboración funcional, desde una elección concreta que haga el usuario mediante las clases visuales, hasta la unidad de información almacenada en la base de datos o en ficheros. Las colaboraciones, por tanto, permiten el trasiego de información desde la parte visible de la aplicación, hasta el núcleo más interno y viceversa.

### 5.4.4. Aplicación

Como ejemplo de aplicación de este patrón se ha elegido uno extraído del módulo de presentación del conocimiento, del caso práctico desarrollado en el anexo. En concreto nos referimos a la construcción de una red de conceptos. La creación de la red de conceptos es un claro ejemplo de colaboración funcional, puesto que tiene una componente visual, la de creación gráfica de la red en sí, y por otro lado tiene un componente funcional, en lo que se refiere a llenar con información dicha red. En él ponemos de manifiesto cómo un evento directo desencadenado por el usuario, como puede ser: '*Construir la red de conceptos*', conlleva un gran número de pequeñas tareas que han de cumplirse para que dicho escenario se cumpla.

La conclusión del refinamiento del modelo de regulación de objetos visuales resulta ser la siguiente:

MEDIANTE LAS PROTEÍNAS SE DESENCADENAN MECANISMOS BIOLOGICOS QUE A SU VEZ ESTAN CONSTITUIDOS POR OTROS PROCESOS  
↔ MEDIANTE LOS OBJETOS VISUALES SE DESENCADENAN ESCENARIOS QUE A SU VEZ ESTAN COMPUESTOS POR UNA SERIE DE FUNCIONES.

## PATRON PARA CREAR UNA RED DE CONCEPTOS

### Objetivo

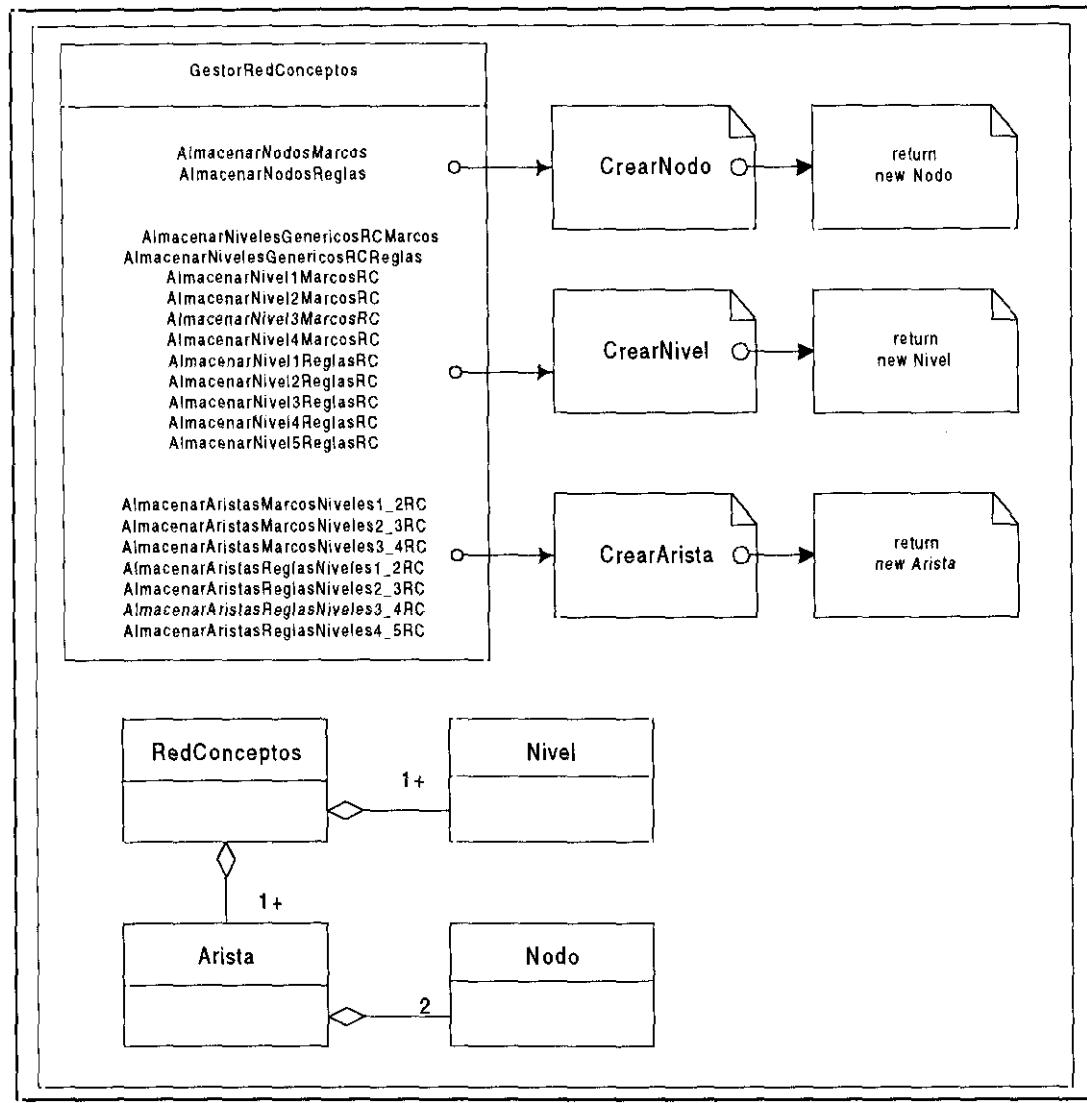
Proporcionar la funcionalidad necesaria para la creación de una red de conceptos como mecanismo de presentación del conocimiento.

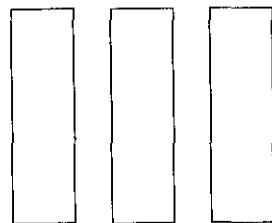
### Participantes

- **GestorRedConceptos**.- Clase que gestiona todo lo relacionado con la elaboración de la red de conceptos.
- **RedConceptos**.- Clase que representa a la red de conceptos como tal.
- **Nivel**.- Clase que representa los niveles de la red de conceptos.
- **Nodo**.- Clase que representa los nodos de la red de conceptos.
- **Arista**.- Clase que representa las aristas de la red de conceptos.

Cada una de estas clases tiene una funcionalidad propia que interviene en la creación de los componentes de la red de conceptos.

### Diagrama





# **ELABORACIÓN DE UN PATRÓN DE ARQUITECTURA**

**PRESENTACIÓN DEL PATRÓN DE  
ARQUITECTURA  
SUBSISTEMA DE DESARROLLO  
SUBSISTEMA DE EJECUCIÓN**



# **Capítulo 6**

---

## **Presentación del patrón de arquitectura**

### **6.1. Planteamiento del problema**

Dado el avance de las nuevas tecnologías, el concepto y utilidad de un generador de sistemas basados en el conocimiento ha variado desde sus orígenes. Actualmente una herramienta de esta índole debe proporcionar diferentes esquemas de representación del conocimiento, así como de propagación del mismo, y ha de ofrecer la posibilidad de introducir manejo de la incertidumbre. Por otro lado las consultas deben admitir recursos multimedia con el fin de ofrecer una solución en el menor tiempo posible, al trabajar con distintos tipos de información. Todas estas necesidades que surgen, dados los cambios protagonizados por la tecnología, han desencadenado el desarrollo de **un patrón de arquitectura** que facilite la construcción de herramientas para la toma de decisiones que esté a la altura de los avances tecnológicos acaecidos.

## 6.2. Requisitos

El patrón de arquitectura propuesto, PARGEN, se cumple en los casos en los que se quiera llevar a cabo el desarrollo de una herramienta para la toma de decisión que cumpla las siguientes características:

- Admitir el conocimiento codificado en forma de marcos, reglas de producción y marcos causales, siendo ésta última la forma cualitativa de introducir el conocimiento que va a constituir una red bayesiana.
- Admitir manejo de la incertidumbre en forma de probabilidad subjetiva, factores de certeza y red bayesiana.
- Admitir recursos multimedia para la elaboración de las consultas del S.B.C.
- Admitir razonamiento en forma de red de conceptos y en forma de árbol de decisión.

Partiendo de estas necesidades iniciales, podemos aplicar PARGEN en la construcción de un generador de S.B.C.

## 6.3. Solución

El sistema a desarrollar es un generador de S.B.C. cuyo objetivo es tomar una decisión a partir de un conocimiento introducido previamente y proporcionar, si es el caso, un valor numérico que justifique la solución a la que se ha llegado.

Proponemos una arquitectura que ilustramos esquemáticamente en la figura 6.1. En dicha arquitectura hemos considerado necesario separar la funcionalidad deseada en dos subsistemas. Un subsistema se identifica por los servicios que proporciona [149], así que, siguiendo esta línea de razonamiento, diferenciamos dos servicios muy claros: el servicio que se refiere a la introducción de la información y el que se refiere a la consulta de la misma, estableciendo respectivamente el subsistema de desarrollo y el subsistema de ejecución.

El subsistema de desarrollo engloba todo lo relativo a la creación de los S.B.C., es decir, a la construcción de la base de conocimiento propiamente dicha, así como su validación y verificación. El subsistema de ejecución está relacionado con la consulta de los S.B.C. Cada uno de los subsistemas está a su vez dividido en subsistemas de más bajo nivel, a los que denominaremos módulos. No olvidemos que el objetivo de un módulo, según Rumbaugh [149] es capturar una particular perspectiva o vista del sistema en que se encuentren.

En la figura que se ilustra a continuación, especificamos con flechas continuas aquellos módulos que son obligatorios en el desarrollo de cualquier sistema basado en el conocimiento, mientras que las flechas discontinuas representan los módulos que son opcionales.

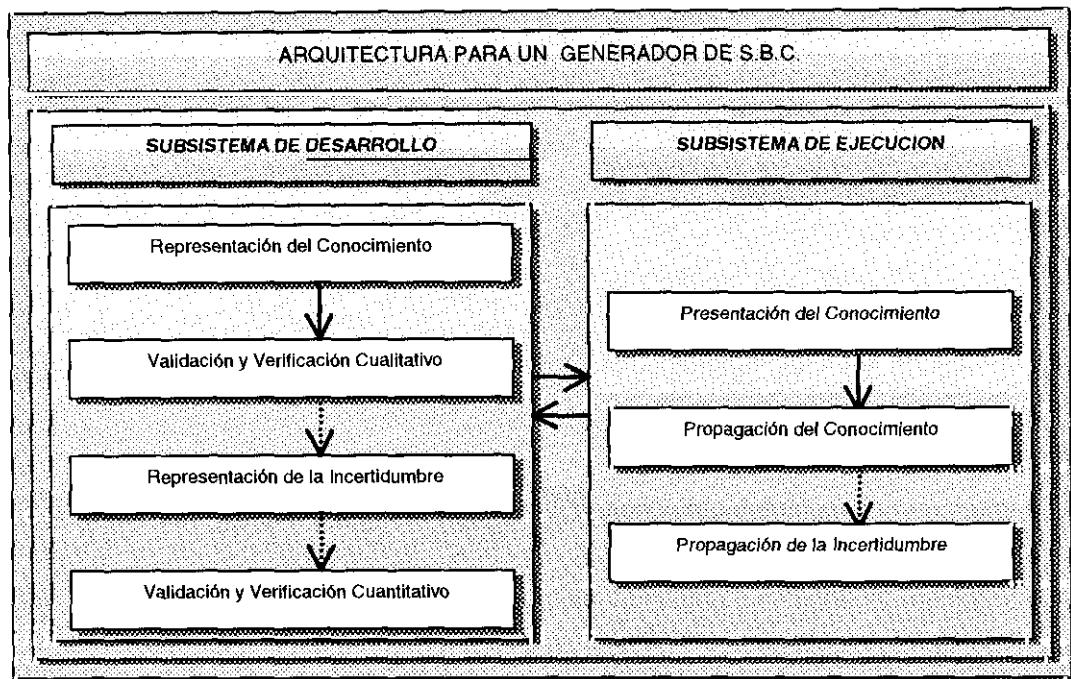


Figura 6.1. Patrón de arquitectura de un generador de S.B.C. (PARGEN)

A continuación pasaremos a describir los módulos en que se han dividido cada uno de los subsistemas anteriores:

a) Subsistema de desarrollo.- Está formado por los módulos que se detallan a continuación.

- Módulo para la representación del conocimiento.- Este módulo está relacionado con la construcción de las bases de conocimiento. Mediante él proponemos una forma sencilla de introducir el conocimiento mediante reglas, marcos de clasificación y marcos causales como unidades. Los marcos de clasificación constituyen los marcos clásicos (frames), los denominamos así para diferenciarlos claramente de los marcos causales.
- Módulo para la representación de la incertidumbre.- Se relaciona con la introducción de la incertidumbre asociada a las unidades de información que forman la base de conocimiento. Este módulo es opcional, y no todas las unidades de información

permiten todos los tipos de manejo de incertidumbre. En el caso de la aplicación propuesta, se proponen tres métodos de introducir la incertidumbre: Probabilidad Subjetiva, Factores de Certeza y Redes bayesianas. Las reglas de producción pueden asociarse a cualquiera de los dos primeros. A los marcos no se les asocia ninguno de los métodos y los marcos causales van ligados a las redes bayesianas.

- **Módulo para la validación y verificación**.- Alude al chequeo de los datos introducidos por el usuario, referentes tanto al conocimiento como a la incertidumbre. De esta forma se establecen dos módulos de validación y verificación: uno cualitativo y otro cuantitativo.

b) **Subsistema de ejecución**.- Constituido por los módulos que se presentan a continuación.

- **Módulo para la presentación del conocimiento**.- Trata del tipo de presentación de los conocimientos que el usuario desea para llevar a cabo la propagación de los mismos. Puede ser en forma de árbol de toma de decisiones o en red de conceptos (Redes SMCPC<sup>4</sup>). Tanto la presentación en árbol de toma de decisiones y en redes de conceptos se puede aplicar tanto a reglas como a marcos de clasificación. Los marcos causales tienen su propia forma de presentación del conocimiento mediante las redes bayesianas.
- **Módulo para la propagación del conocimiento**.- Se encarga del proceso de inferencia en sí. Este módulo proporciona al experto la posibilidad de poder iniciar la inferencia con la información que él mismo disponga, con el fin de poner en marcha el motor de inferencia. En caso de no tener ninguna información, el motor de inferencia, mediante diferentes mecanismos que ya explicaremos más adelante, comienza a hacer preguntas al usuario, y combina las respuestas facilitadas con la información almacenada en la base de conocimiento. Cuando el sistema llega a una solución, el motor de inferencia se detiene, esperando una nueva orden por parte del usuario.
- **Módulo para la propagación de la incertidumbre**.- Se relaciona con la propagación de la incertidumbre. Existen tres opciones de manejo de la incertidumbre en el sistema propuesto. Una de ellas es la de la probabilidad subjetiva, el mismo método de propagación que se introdujo en PROSPECTOR, otra es la de los factores de certeza, que equivale al introducido en MYCIN y por último las redes bayesianas, que reflejan un método probabilístico causal.

---

<sup>4</sup> 'Redes Semánticas de Conceptos, Proposiciones y Conocimientos'; comunicación personal a cargo de M.Levy

# **Capítulo 7**

---

## **Subsistema de desarrollo**

### **7.1. Representación del conocimiento**

El módulo para la representación del conocimiento que proponemos presenta tres posibles formas de introducir el conocimiento: reglas de producción, marcos de clasificación y marcos causales. Las reglas de producción y los marcos de clasificación se utilizan mucho en los S.B.C. para formalizar el conocimiento experto. Por otro lado los marcos causales, son una propuesta del autor para representar el conocimiento que construye una red bayesiana.

Se presenta pues un módulo mixto para la representación, con tres posibles formas de introducir el conocimiento y que son independientes entre sí. La razón por la que nos hemos planteado estos tres esquemas ha sido, porque hay problemas que se ajustan más a una lógica de primer orden (reglas), otros se ajustan más a patrones (marcos) y otros más a una solución causal (marcos causales); este módulo para la representación nos va a proporcionar soluciones de formalización para diferentes tipos

de problemas. Por ejemplo un problema de taxonomía o clasificación, se recomienda resolverlo empleando marcos de clasificación como mecanismo de representación. Un problema que responde a la emisión de un diagnóstico a partir de una serie de observaciones se debe plantear con reglas de producción. Por otro lado, un problema que responda a un esquema causal de comportamiento, se puede formalizar con los marcos causales.

### 7.1.1. Representación con reglas de producción

La forma de representación más comúnmente elegida por los S.B.C. son las reglas de producción. Las reglas de producción proporcionan un esquema de planteamiento lógico, que está de acuerdo con la forma que tiene el experto de expresar su conocimiento. No hemos de olvidar que el experto trata de establecer unas reglas generales de comportamiento a partir de la observación de una serie de casos particulares. Las reglas son pues, la forma de representación con que se encuentra más familiarizado. Por otro lado, cuando el experto elige construir un S.B.C. empleando las reglas de producción, no solamente va a tener que preocuparse de formalizar su conocimiento como reglas, sino que van a existir otras dos unidades de conocimiento, fundamentales para elaborar sus reglas: los atributos y los objetos.

Los atributos son realmente propiedades que se pueden asignar a cualquiera de los elementos de su área de conocimiento. Pueden tomar valores que se introducen en el sistema mediante el editor de atributos, de éste y del resto de los editores se hablará en el capítulo 10, y obedecen a un tipo determinado, dependiendo de los valores que se les vaya a asignar. Están formados por un nombre, una descripción, el tipo a que obedecen y el rango dentro de éste.

El esquema general se presenta en la tabla 7.1.

<b>Nombre</b>
<b>Descripción</b>
<b>Tipo</b>
<b>Rango</b>

Tabla 7.1. Esquema de representación de un atributo

Los objetos, cuyo esquema general obedece a lo que se conoce como objeto en O.O., están definidos por un conjunto de propiedades que toman un determinado valor. Los objetos se introducen en el sistema mediante el editor de objetos y están formados por el nombre, la descripción y un conjunto de una o varias ternas: atributo/operador relacional/valor. El esquema general se muestra en la tabla 7.2.

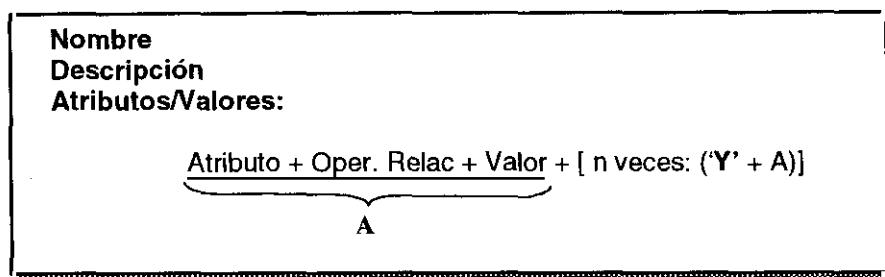


Tabla 7.2. Esquema de representación de un objeto

Las reglas de producción conjugan, entre otros, los dos elementos anteriormente detallados. Se introducen en el sistema mediante el editor de reglas. Su estructura se detalla en la tabla 7.3, y a continuación explicamos brevemente las partes de que consta:

- a) Etiqueta inicial o Nombre.- Nombre que se quiere dar a la regla en cuestión.
- b) Antecedente.- Formado por ‘n’ premisas o líneas de antecedente, que realmente son ternas del tipo: ‘Atributo/Operador relacional/Valor’ o bien ‘Objeto/Operador relacional/Valor’. Estas se presentan unidas entre sí por operadores lógicos, que puede ser: ‘Y’ y ‘O’ y pueden estar agrupadas mediante paréntesis. Una premisa puede ocasionalmente estar formada por la negación o afirmación de una hipótesis, lo que querrá decir que mediante una línea se está negando o afirmando todo el conjunto de premisas de la regla que contiene dicha hipótesis.
- c) Hipótesis.- Meta que se cumple en caso de satisfacer el antecedente completo.
- d) Consecuente.- Formado por ‘n’ líneas cuya estructura puede estar constituida opcionalmente por líneas tipo: ‘Instrucción/Valor’, ‘Atributo/Operador relacional/Valor’ o bien ‘Objeto/Operador relacional/Valor’, unidas por operadores tipo: ‘Y’. Una instrucción es una orden del sistema, por ejemplo abrir, ejecutar o mostrar, entre otras, y el valor es un fichero de texto, un ejecutable o un fichero gráfico, respectivamente.

Una novedad que presentan las reglas de producción del sistema propuesto es la facilidad de ir viendo en la pantalla lo que se va introduciendo, *wysiwyg* (*What You See Is What You Get*), de forma que a medida que se van introduciendo las reglas, se puede ver su contenido, diferenciándose los elementos de que consta mediante un código de colores. Una regla recién introducida, al intentar grabarse, desencadena todos los mecanismos de validación que comprueban que la regla introducida es correcta. Si no lo fuera, se dispone de un editor de errores posibles y localización de los mismos, para que la corrección sea lo más inmediata posible.

Por otro lado, una vez introducidas todas las reglas en la base de conocimiento, se comprueba la consistencia de la base de conocimiento completa. El proceso es similar al que ocurría para una unidad de conocimiento. Se dispara el módulo para la verificación y cualquier error que pueda haber, es mostrado en el editor de errores con el objetivo de que su corrección sea sencilla.

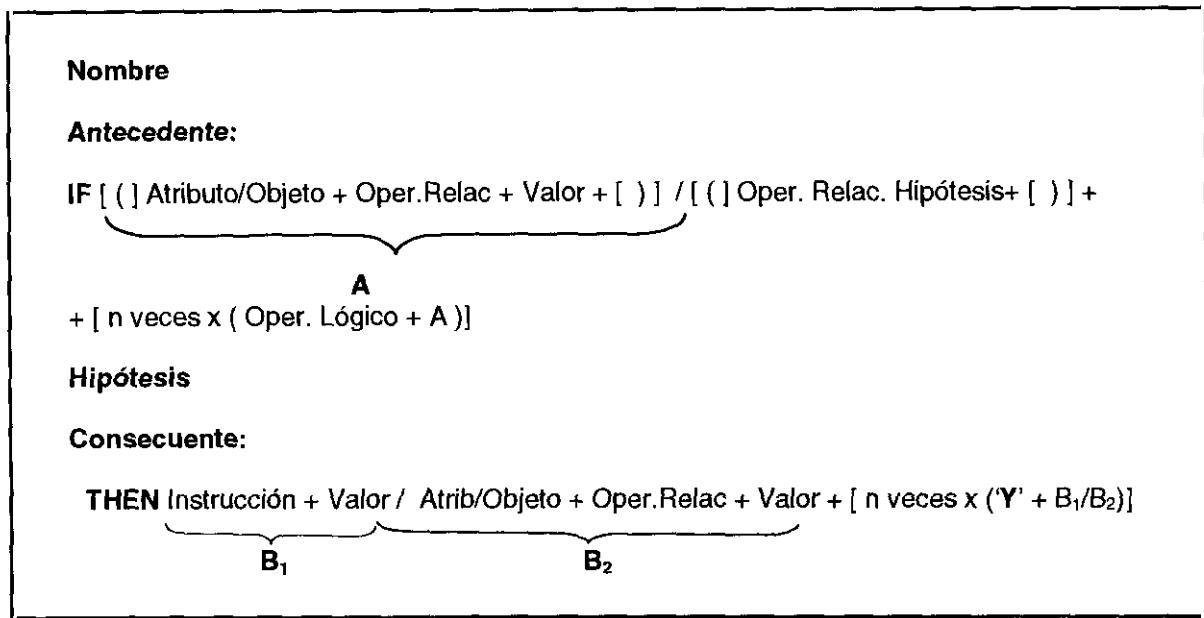


Tabla 7.3. Esquema de representación de una regla de producción

### 7.1.2. Representación con marcos de clasificación

Los marcos de clasificación responden a los llamados tradicionalmente *frames*. El editor de los marcos de clasificación del caso práctico presentado es muy sencillo de manejar. En la tabla 7.4. se presenta el esquema general al que responde un marco de clasificación para el sistema propuesto.

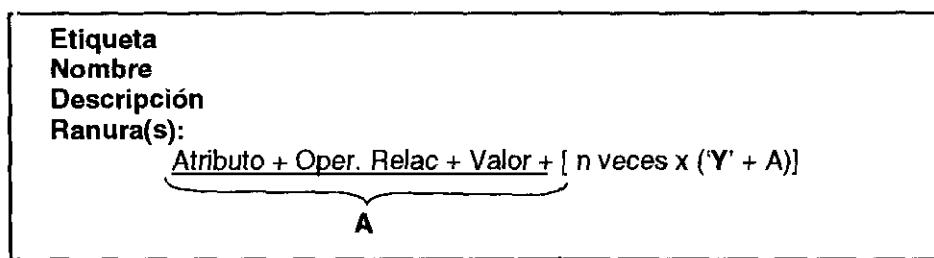


Tabla 7.4. Esquema de representación de un marco de clasificación

Un marco de clasificación está formado por una etiqueta, un nombre, una descripción y un conjunto de ternas: ‘atributo/operador relacional/valor’, que reciben el nombre de ranuras, *slots*. Al nombre de un marco de clasificación lo llamaremos, a partir de ahora, meta.

Las bases de conocimiento basadas en marcos de clasificación tienen también una validación y una verificación, la primera de cada unidad por separado, y la segunda que comprueba la consistencia de toda la base de conocimiento al completo.

### 7.1.3. Representación con marcos causales

El uso de las redes bayesianas como mecanismo mixto de representación del conocimiento y propagación de la incertidumbre, resulta bastante cómodo en un primer momento, al integrar en una sola estructura, información cualitativa, cuantitativa y causal, pero su estructura fija hace que, a la hora de hacer modificaciones, sea necesaria la actuación de un ingeniero del conocimiento para que el modelo resultante después de dichas modificaciones, sea consistente y no contenga redundancias.

La creación de un editor particular previo a la construcción de la red bayesiana, que favorezca al usuario la introducción de la información causal, permite estructurar el conocimiento de una forma ordenada y coherente, siendo después la propia herramienta la que genera a partir de la información causal introducida, la red causal.

Las redes bayesianas pues, aunque muy aclaratorias en cuanto a la organización de todo el dominio del conocimiento y al manejo de la incertidumbre, presentan ciertas deficiencias a la hora de introducir nuevo conocimiento o modificar el ya existente. Así surgen intentos de completar este aspecto introduciendo una interfaz de usuario o adaptando pequeños módulos a la red que lo que hagan sea recoger la información de la red bayesiana y mostrarla al usuario de una forma mucho más fácil de entender, como es el caso de BANTER [74].

Siguiendo esta línea de razonamiento, hemos considerado adecuado añadir, previa a la creación de la red causal, una pantalla para introducir el conocimiento causal extraído del problema, proponiendo como forma de representación los marcos causales, cuyo esquema general se muestra en la tabla 7.5.

<b>Efecto</b>	
<b>Descripción</b>	
<b>Causa(s):</b>	
	Atributo + Valor + [ n veces x ('Y' + A)]

Tabla 7.5. Esquema de representación de un marco causal

De esta forma, aunque el esquema parece igual al de los marcos de clasificación, existe una diferencia importante. En los marcos de clasificación, las ranuras eran un agregado de tres unidades independientes entre sí: atributo, operador relacional y valor. En el caso de los marcos causales, las causas son agregados de dos elementos: un atributo y un valor.

## 7.2. Validación y verificación de la base de conocimiento

La validación y verificación de una base de conocimiento son necesarias para la correcta propagación del conocimiento por parte del motor de inferencia. Por un lado la validación asegura el chequeo sintáctico, semántico y de coherencia de cada unidad de conocimiento, y por otro la verificación asegura el chequeo de coherencia de toda la base de conocimiento. La presencia de este módulo en un S.B.C. hace más sencillo el posterior mantenimiento del S.B.C.

### 7.2.1. Aspectos generales

El principal objetivo del módulo para la validación y verificación en el caso práctico propuesto – *a partir de ahora, MVVBC* – es actuar sobre la base de conocimiento comprobando toda la información suministrada, con el fin de que la posterior introducción de la incertidumbre sea consistente y que el motor de inferencia actúe, en cualquier caso, sobre unos conocimientos bien construidos. Este módulo garantiza una correcta base de conocimientos tanto en el ámbito estructural como funcional.

Debido al carácter mixto de la base de conocimiento, al contemplar tres tipos diferentes de unidades de conocimiento, hemos diferenciado el comportamiento del MVVBC según se trate de reglas de producción, marcos de clasificación o marcos causales. Por otro lado también hemos diferenciado el punto de vista de desarrollo de la base de conocimiento y el punto de vista de explotación [205], estableciendo diferentes comprobaciones en uno y en otro caso, que veremos a continuación, al analizar la estructura interna.

### 7.2.2. Estructura interna

La estructura interna del MVVBC consta de un conjunto de analizadores que realizan una serie de operaciones. Estas analizadores aseguran su consistencia tanto a nivel del correcto empleo del orden sintáctico de los datos, como al nivel de la más exacta utilización de su significado. Y aseguran igualmente que no existan redundancias en la información, no sólo en el orden individual de cada

unidad de conocimiento – regla, marco de clasificación o marco causal – sino a nivel del conjunto de todas ellas. Nos ha parecido adecuada la diferenciación entre validación y verificación que proponía Preece en 1992 y a la que aludíamos en el capítulo 2, por lo tanto la validación la realizaremos teniendo en cuenta la unidad de conocimiento aislada, y la verificación considerando la unidad de conocimiento en relación con el resto de unidades que componen la base de conocimiento.

Así, puesto que una base de conocimiento presenta dos niveles de refinamiento: el que se aplica en desarrollo, que se relaciona con una vista estática de la misma y el que se aplica en ejecución, que se relaciona con una vista dinámica, podemos afirmar que los analizadores que actúan sobre unidades de conocimiento individuales son los relacionados con las propiedades estáticas de la base de conocimiento, mientras que los analizadores que actúan sobre toda la base en su conjunto, son los relacionados con las propiedades dinámicas. Este término de propiedades estáticas y dinámicas ya era introducido por Preece en 1997.

La estructura interna del MVVBC varía según la codificación de conocimientos que hemos elegido, teniendo en cuenta que la flexibilidad permitida por el sistema a la hora de introducir el conocimiento, es mayor o menor según se trate de una base apoyada en reglas o basada en marcos causales o de clasificación, respectivamente.

Destacamos por su importancia, que el MVVBC consta de un conjunto de analizadores que actúan como controladores de diversos aspectos de las unidades de conocimiento, en el momento en que se van introduciendo en el sistema. Estos aspectos se resumen en la sintaxis, el significado y las redundancias parciales y globales. En la figura 7.1 se muestra la estructura interna del MVVBC, con cada uno de sus analizadores y las propiedades que caracterizan a cada uno. Y a continuación pasamos a enumerar y describir cada uno de ellos.

- a) Analizador sintáctico
- b) Analizador semántico
- c) Analizador de coherencia parcial
- d) Analizador de coherencia total
  
  
  
  
  
  
- a) Analizador sintáctico.- Es el controlador que se encarga de operar sobre las unidades de conocimiento introducidas comprobando que su sintaxis sea correcta. De esta forma no será igual la función que realice el analizador sintáctico sobre una regla, sobre un marco de clasificación o sobre un marco causal. La creación del analizador sintáctico asegura que una base de conocimiento cumpla una serie de propiedades que se exponen a continuación:

- Compleitud.- La unidad de conocimiento figurará completa, es decir, estará formada por todos los elementos que se entiendan como mínimos para considerarla una unidad de conocimiento del tipo elegido: regla de producción, marco de clasificación o marco causal.
  - Unicidad.- La unidad de conocimiento estará formada por elementos únicos, no repetidos.
  - Equivalencia.- La unidad de conocimiento se ajustará al patrón genérico de la misma.
  - Anidamiento.- La unidad de conocimiento, si consta de varias cláusulas unidas entre sí por operadores, no presentará desajustes de anidamiento en relación con los paréntesis de que conste.
  - Exclusividad.- La unidad de conocimiento debe darse de forma exclusiva y con carácter permanente en la propia base de conocimiento.
- b) Analizador semántico.- Es el controlador que va a garantizar que no existen incorrecciones en el ámbito de significado entre los distintos elementos que compongan la unidad de conocimiento analizada. Otro de sus objetivos es no permitir contradicciones en una unidad determinada, ni errores de incompatibilidad de tipos. Consta de las propiedades que se detallan a continuación.
- Existencia previa.- Cuando en una unidad concreta de conocimiento se mencionan otras subunidades de conocimiento, éstas deben haber sido creadas previamente.
  - Concordancia.- Esta propiedad hace alusión tanto en cuanto a lo referente a los tipos de datos, como a lo referente a afirmaciones o negaciones que figuren en una determinada unidad de conocimiento.
- c) Analizador de coherencia parcial.- Se presenta como el controlador encargado de asegurar que no existen redundancias entre las subunidades que forman la unidad de conocimiento en estudio. La propiedad por la que viene definido es la siguiente:
- Redundancia.- Se asegura que no existan repeticiones de subunidades en el ámbito interno de la unidad de conocimiento concreta.

Hasta ahora hemos descrito tres analizadores cuya función de control se realiza sobre cada unidad de conocimiento a medida que van siendo introducidas por el usuario.

A continuación describimos el único analizador del MVVBC que actúa sobre la base de conocimiento total. Comenzamos pues las validaciones y verificaciones dinámicas del sistema.

d) Analizador de coherencia total.- Es el controlador encargado de evitar repeticiones o redundancias inútiles a lo largo de toda la base de conocimiento. Consta de una serie de propiedades que se indican a continuación:

- Unicidad- La unidad de conocimiento será única e irrepetible.
- Secuencialidad o Circularidad- Las unidades de conocimiento, cuando se presenten encadenadas entre sí, no formarán bucles, sino que se ajustarán a un esquema lineal de encadenamiento.

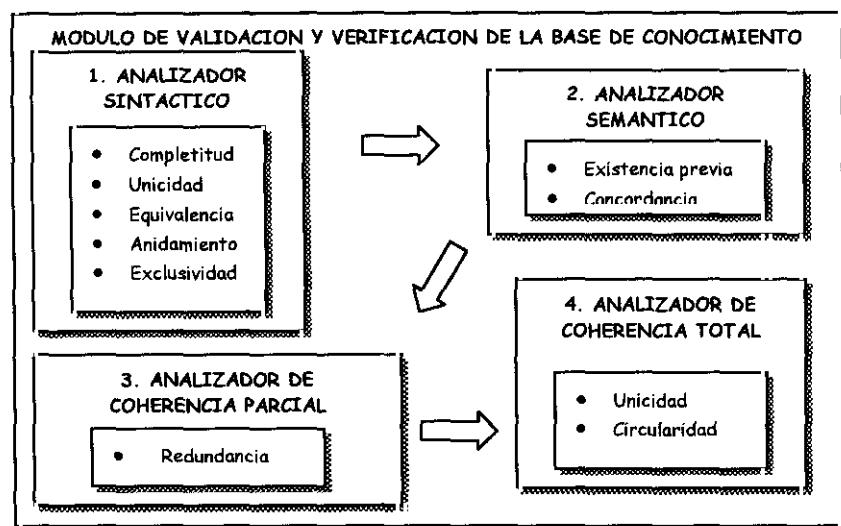


Figura 7.1. Estructura interna del módulo de validación y verificación

### 7.2.3. Funcionamiento en reglas de producción

Para un sistema basado en reglas de producción, ha de tenerse presente que, si en el mismo fuese introducida una regla de producción, invariablemente entraría en funcionamiento el módulo para la coherencia para evitar inconsistencias posteriores antes de proceder a su grabado en la base de datos.

Para entender el funcionamiento del módulo para la coherencia, en el caso de las reglas de producción, debemos tener presente la estructura interna de una regla.

Los datos pueden ser introducidos en el sistema manualmente, para lo cual proponemos una pantalla en la que aparece la composición de la regla representada mediante un código de colores, con sus elementos debidamente diferenciados.

El usuario dispone también de todos los elementos de la regla en la misma pantalla separadas mediante cajas de texto individuales, para que, mediante un simple arrastre de ratón, pueda escribir toda una regla, con la excepción de los valores que tomen los atributos/objetos, que deberán ser introducidos mediante teclado. Una vez terminada la regla, y previo a la grabación, invariablemente tendrá lugar la puesta en marcha del módulo para la coherencia que, en este caso, estará formado por los cuatro analizadores explicados anteriormente.

a) Analizador sintáctico.- El analizador sintáctico propuesto está formado por un conjunto de operaciones que se consideran imprescindibles para preservar la sintaxis de cada una de las reglas introducidas en el sistema. Estas operaciones vienen determinadas por un conjunto de propiedades que se describen a continuación.

- Completitud de la regla.- La regla tiene completos sus elementos imprescindibles, es decir, contiene un nombre, al menos una línea de antecedente y una hipótesis.
- Unicidad de la cabecera.- La cabecera de la regla no está repetida.
- Equivalencia del antecedente y consecuente con el patrón general.- Se cumple el esqueleto general de antecedente y consecuente, que se presentó en la tabla 7.3.
- Anidamiento correcto.- El número de paréntesis es correcto y su distribución en la regla es el adecuado.
- Unicidad de la hipótesis.- La hipótesis de la regla no está repetida.
- Exclusividad de la hipótesis.- La hipótesis de la regla no se encuentra almacenada como objeto.

b) Analizador semántico.- El analizador semántico propuesto está formado por un conjunto de operaciones imprescindibles para preservar la concordancia semántica de cada una de las reglas introducidas en el sistema. Estas operaciones se describen a continuación:

- Existencia previa de objetos y atributos.- Los objetos y atributos a que hagan alusión las reglas, deben estar ya creados con los editores correspondientes.
- Concordancia de tipos.- Se mantiene la coherencia de tipos, entre Atributo/Operador/Valor e Instrucción/Valor.
- Concordancia entre antecedente y consecuente.- No existe contradicción entre las asignaciones/desasignaciones de atributos/Objetos del consecuente, con respecto a los que ya existían en el antecedente.

- Concordancia entre antecedente y objetos.- No existe contradicción entre los Atributos/Valores del antecedente, con los Atributos/Valores de cualquier objeto presente en dicho antecedente
- c) Analizador de coherencia parcial.- El analizador de coherencia parcial se encarga de comprobar la coherencia de cada regla actuando sobre ella; es necesario que no existan redundancias ocultas, ya que resulta frecuente encontrarse con éstas enmascaradas en los operadores lógicos y en los paréntesis, y por tanto, redundancias no visibles a simple vista sino que requieran analizar el contenido de la regla. En algunas ocasiones, dos líneas de antecedente que en principio pueden ser redundantes, al operar sobre la regla, puede resultar que estando separadas por operadores 'O' y unidas a su vez con otras premisas mediante operadores 'Y', lo que era una redundancia en principio resulta ser una regla correcta al final. El analizador se encarga de resolver este tipo de estructuras que a simple vista podrían no verse.
- Redundancia.- Asegura que no existan redundancias en el ámbito interno de cada regla.
- d) Analizador de coherencia total.- El analizador de coherencia total tiene como objetivo evitar redundancias en el conjunto de las reglas de la base de conocimiento. En síntesis ha de cumplirse:
- Unicidad de la regla.- No existen dos reglas con todos sus elementos idénticos.
  - Secuencialidad.- No existen bucles en el encadenamiento de las reglas. Cuando dos reglas estén unidas entre sí porque en el antecedente de la segunda figure la hipótesis de la primera, el consecuente de la segunda no debe encadenarse de nuevo con la primera regla por medio de alguna de las afirmaciones del consecuente, puesto que se formaría un bucle, produciendo el bloqueo del sistema e impidiendo como consecuencia proseguir hacia delante en el proceso de inferencia.

#### 7.2.4. Funcionamiento en marcos de clasificación

El módulo para la coherencia de una base de conocimiento basada en marcos de clasificación resulta mucho más sencillo si lo comparamos con los módulos de coherencia aplicados sobre bases de conocimiento basadas en reglas o en marcos causales. Los marcos de clasificación los introduce el usuario con un simple arrastre de ratón, debido a que todos los elementos de que consta un marco se presentan en el editor correspondiente, salvo los valores de las ranuras, que se introducen mediante teclado.

En el caso de bases de conocimiento basadas en marcos de clasificación resulta obvio que las operaciones a realizar por el analizador sintáctico van a ser menos numerosas que las que han surgido en el apartado anterior. El analizador semántico se ocupa de mantener la coherencia en el interior del marco causal y los analizadores de coherencia parcial y total se encargan de mantener la coherencia interna y global de toda la base de conocimiento.

a) Analizador sintáctico.- El analizador sintáctico asegura una sintaxis correcta antes de almacenar los marcos. Las operaciones de que consta este analizador son las siguientes:

- Completitud.- El marco tiene como mínimo una etiqueta, un nombre y una ranura.
- Unicidad de la etiqueta y nombre.- Tanto la etiqueta del marco como el nombre, son únicos e irrepetibles.

b) Analizador semántico.- El analizador semántico preserva la coherencia semántica de cada uno de los marcos de clasificación introducidos en el sistema. A continuación se exponen las propiedades que debe cumplir:

- Concordancia de tipos.- Se mantiene la coherencia de tipos, entre Atributo/Operador/Valor.
- Concordancia entre ranuras.- No existe contradicción entre las asignaciones/desasignaciones de las diferentes ranuras de un marco de clasificación.

c) Analizador de coherencia parcial.- El analizador de coherencia parcial asegura que cada marco de clasificación dispone de sus elementos mínimos para que pueda ser grabado en la base de conocimiento. Este analizador consta de la siguiente operación:

- Redundancia.- Se comprueba que no existan ranuras repetidas en el marco, ni siquiera de sus atributos por separado, aunque tengan diferentes valores: sería contradictorio.

d) Analizador de coherencia total.- El analizador de coherencia total, tiene en cuenta las operaciones a realizar considerando la base de conocimiento en su conjunto.

- Unicidad de los marcos.- No puede haber marcos iguales.

### 7.2.5. Funcionamiento en marcos mausales

El módulo para la coherencia, en el caso de una base de conocimiento de marcos causales, consta de cuatro analizadores, siendo también de funcionamiento algo simplificado si se compara con el que utiliza reglas de producción.

- a) Analizador sintáctico.- El analizador sintáctico asegura que cada marco causal dispone de sus elementos mínimos para que pueda ser guardado en la base de conocimiento. Las operaciones de que consta este analizador son las siguientes:
  - Completitud.- El marco causal tiene como mínimo una etiqueta y una causa.
  - Unicidad de la etiqueta.- La etiqueta del marco causal es única e irrepetible.
- b) Analizador semántico.- El analizador semántico preserva la coherencia semántica de cada uno de los marcos causales introducidos en el sistema. Consta de las siguientes propiedades:
  - Concordancia entre causas.- No existe contradicción entre las asignaciones/desasignaciones de las diferentes causas de un marco causal.
- c) Analizador de coherencia parcial.- El analizador de coherencia parcial asegura que cada marco causal dispone de unos elementos mínimos para que pueda ser grabado en la base de conocimiento. La operación que se considera es únicamente la siguiente:
  - Unicidad.- Se comprueba que no existen causas repetidas en el marco: sería contradictorio.
- d) Analizador de coherencia total.- El analizador de coherencia total tiene en cuenta propiedades a realizar contando con la base de conocimiento en su conjunto. Las operaciones de que consta son las siguientes:
  - Unicidad de los marcos causales.- No puede haber marcos causales iguales.
  - Secuencialidad.- No se admiten ciclos en el conjunto de todos los marcos causales, puesto que el proceso de propagación va a ser el método de mensajes, válido sólo para poliárboles.

## 7.3. Representación de la Incertidumbre

El caso práctico propuesto dispone de tres modos de representación de la incertidumbre: probabilidad subjetiva, factores de certeza y redes bayesianas. Cada uno de estos modos es exclusivo de una unidad de conocimiento concreta, así tanto la probabilidad subjetiva como los factores de certeza son propios de las reglas de producción y las redes bayesianas son aplicables a los marcos causales. A los marcos de clasificación no les hemos asociado ningún modo de representación de la incertidumbre.

### 7.3.1. Aspectos generales

El funcionamiento general del módulo para la representación de la incertidumbre varía de unos casos a otros. En el caso de la probabilidad subjetiva, se hace necesaria la transformación previa de las unidades de conocimiento, de manera que primero se pone en funcionamiento un analizador que transforma las unidades complejas de las reglas en subunidades más simples. Y por otro lado se muestra gráficamente el resultado de dicha transformación.

La representación de la incertidumbre mediante factores de certeza, requiere tan sólo la asociación de dichos factores a las reglas una vez que se han construido, pero no lleva asociado ni el desencadenamiento de procesos que transformen las reglas en subunidades, ni la elaboración gráfica de las unidades de conocimiento con los factores asociados.

En el caso de elegir las redes bayesianas, no se necesita la transformación de las unidades de conocimiento, ya que los marcos causales presentan la forma idónea para que se les asocie probabilidades directamente, por su estructura causal, lo que sí se precisa en este método es una representación gráfica de los marcos causales, para que la introducción de valores se haga más sencilla y se observen a simple vista las relaciones causales existentes.

### 7.3.2. Funcionamiento con probabilidad subjetiva

Las fases por las que consideramos que ha de pasar una base de conocimiento para que pueda admitir probabilidad subjetiva son las que detallamos a continuación:

- a) Creación de una Base de Conocimiento
- b) Actuación del Módulo para la Validación y Verificación
- c) Actuación del Analizador Lógico

- d) Actuación del Analizador Gráfico
- e) Actuación del Analizador probabilístico
- f) Actuación del Analizador de Consistencia

- a) Creación de una Base de Conocimiento.- Se trata de llevar a cabo la construcción de la base de conocimiento basada en Reglas de Producción y Objetos. Este tema ya se comentó en el apartado 7.1.1.
- b) Actuación del Módulo para la Validación y Verificación.- Se trata de validar la base de conocimiento construida anteriormente, este tema ya se expuso en el apartado 7.2.3.
- c) Actuación del Analizador Lógico.- El analizador lógico es el encargado de transformar las reglas de producción que ha introducido el usuario y que ya han pasado por el módulo de validación y verificación. Las reglas introducidas por el usuario están formadas por paréntesis y operadores lógicos ‘Y’ y ‘O’ y se van a transformar en reglas hijas, sin paréntesis y con operadores ‘Y’ únicamente, aplicando la propiedad distributiva. Para ello hemos elaborado un analizador cuyo cometido se desdobra en las siguientes etapas:

c.1) Codificación del antecedente:

- Eliminación de los espacios entre los elementos del antecedente (separar cada elemento por 1 espacio)
- Transformación de cada premisa (objeto/operador relacional/valor // atributo/operador relacional/valor) en una letra del alfabeto.

c.2) Recogida de las expresiones con mayor nivel de anidamiento.

c.3) Aplicación de la propiedad distributiva a cada una de dichas expresiones, si puede aplicarse.

- Eliminación de los paréntesis que rodeaban dicha expresión
- Repetición de este proceso hasta que hayan desaparecido los paréntesis del antecedente

c.4) Descodificación del antecedente

- Recuperación de las premisas iniciales.

d) Actuación del Analizador Gráfico.- El Analizador Gráfico se encarga de representar de manera visual, por un lado los objetos desdoblados en sus ternas atributo/operador/valor y por otro lado las reglas extraídas en el analizador lógico desdobladas en cada una de sus premisas. Las etapas del analizador gráfico son las siguientes:

d.1) Leer la información asociada al elemento a representar; en caso de que sea un objeto, se leerá el número de ternas que contenga y la información asociada a los mismos. En caso de que sea una regla, se leerá el número de subreglas que han aparecido como consecuencia de aplicar el analizador lógico y la información asociada a las mismas.

d.2) Dividir en celdas el área de la pantalla donde se va a mostrar la información gráfica.

d.3) Dibujar el elemento elegido: objeto o regla, colocado en las coordenadas destinadas para ello.

e) Actuación del Analizador Probabilístico.- El analizador probabilístico se encarga de organizar los datos de la base de conocimiento, para que sea sencilla la introducción de las probabilidades por parte del usuario. Se distinguen dos funciones bien diferenciadas y que agruparemos como Analizador Probabilístico I y Analizador Probabilístico II. Por un lado es necesario introducir las probabilidades a priori, para lo cual se pone en marcha el Analizador Probabilístico I, y por otro se introducen las medidas de suficiencia y necesidad, para lo cual se emplea el Analizador Probabilístico II.

El Analizador Probabilístico I consta de las siguientes etapas:

- Extracción de todos los objetos de la base de conocimiento.
- Extracción de todas las hipótesis de las reglas de producción.
- Presentación de ambos tipos de elementos al usuario.
- Asignación de probabilidades a priori a cada uno de los objetos e hipótesis de la base de conocimiento.

El Analizador Probabilístico II consta de las siguientes etapas:

- Extracción de las ternas de cada uno de los objetos de la base de conocimiento.
- Extracción de las premisas de las reglas de producción, una vez que han pasado por el analizador lógico.
- Presentación de ambos tipos de elementos al usuario.

- Asignación de medidas de necesidad y de suficiencia a cada uno de los elementos extraídos.
- f) Actuación del Analizador de Consistencia.- El analizador de consistencia se encarga de validar que las probabilidades introducidas por el usuario estén dentro de los rangos permitidos y que, además, no exista ningún elemento que no tenga su probabilidad asociada. Otra función de este analizador consiste en grabar las probabilidades asociadas una vez que se han validado. El analizador de consistencia consta de las siguientes etapas:
- f.1) Comprobar la consistencia de las probabilidades a priori asociadas a objetos y a hipótesis.
  - f.2) Comprobar la consistencia de las medidas de suficiencia y de necesidad asociadas a las ternas atributo/operador relacional/valor de los objetos y a las premisas de las reglas de producción.
  - f.3) Grabar en base de datos las probabilidades y las medidas, en caso de que los valores sean correctos.

En la figura 7.2. se muestra el funcionamiento genérico para el analizador probabilístico, plasmando de forma esquemática cada una de las etapas del analizador probabilístico I, para las probabilidades a priori, y las etapas del analizador probabilístico II, para las medidas de suficiencia y necesidad. Una vez cumplidas esta serie de fases, el sistema se encuentra preparado para llevar a cabo la propagación de las probabilidades subjetivas, tema que explicaremos en el capítulo siguiente.

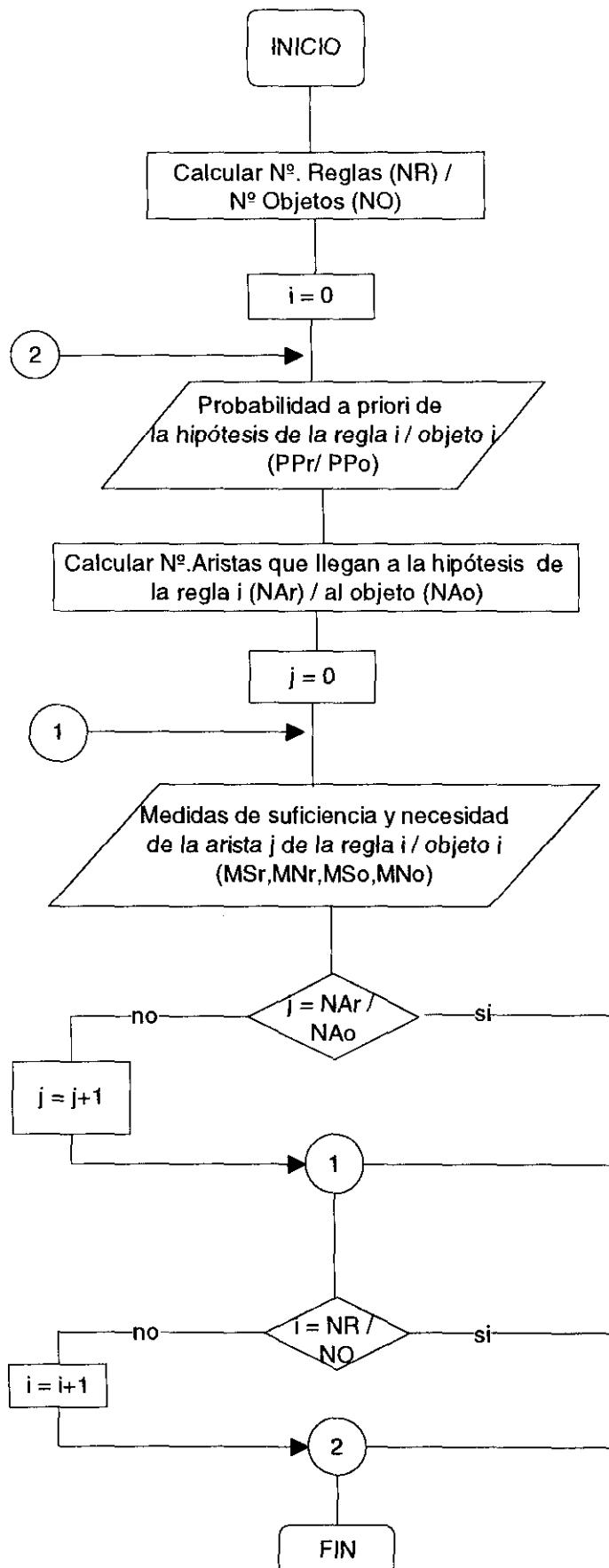


Figura 7.2. Funcionamiento del analizador probabilístico de probabilidad subjetiva

### 7.3.3. Funcionamiento con factores de certeza

Las fases por las que ha de pasar una base de conocimiento construida para admitir factores de certeza son las siguientes:

- a) Creación de una Base de Conocimiento
- b) Actuación del Módulo para la Validación y Verificación
- c) Actuación del Analizador de Certeza
- d) Actuación del Analizador de Consistencia

Una vez cumplidas esta serie de fases, el sistema se encuentra preparado para llevar a cabo la propagación de los factores de certeza, como se detallará en el módulo siguiente.

- a) Creación de una Base de Conocimiento basada en Reglas de Producción y Objetos.- Este tema ya se ha comentado en el apartado 7.1.1.
- b) Actuación del Módulo para la Validación y Verificación, de la Base de Conocimiento.- Este tema ya se ha comentado en el apartado 7.2.3.
- c) Actuación del Analizador de Certeza.- Este analizador es el encargado de facilitar la entrada de los factores de certeza cada vez que se introduce una regla nueva.
- d) Actuación del Analizador de Consistencia.- Este analizador es el encargado de verificar la consistencia de los valores numéricos introducidos en cada regla.

### 7.3.4. Funcionamiento con redes bayesianas

Como se ha apuntado anteriormente, el caso práctico propuesto ofrece la posibilidad de introducir, tanto información cuantitativa como cualitativa, mediante una serie de pantallas que hemos construido y, en un proceso posterior, se procede a elaborar la red bayesiana. De esta forma, el usuario no construye directamente la red bayesiana, sino que es la propia herramienta quien lo hace, pasando previamente por el MVVBC con el fin de que la red creada sea consistente.

Todo este proceso se puede resumir en un conjunto de fases que se detallarán a continuación. Dichas fases, van a requerir al principio, por parte del experto, haber llevado a cabo labores de ingeniería del conocimiento, con el fin de que la introducción de la información no se prolongue demasiado en el tiempo y no dé lugar a incoherencias. Ya Spiegelhalter, en 1993, determinaba tres

fases para la elaboración de las redes bayesianas: fase cualitativa, fase probabilística y fase cuantitativa.

En este caso, durante la fase de desarrollo, la elección de un S.B.C. que funcione como red bayesiana, conlleva pasar por cuatro fases: fase cualitativa, fase de validación y verificación, fase gráfica y fase cuantitativa.

Las dos primeras fases ya se han detallado en los apartados 7.1.3. y 7.2.5. y hacen alusión a la representación de los marcos causales y al módulo para la validación y verificación, respectivamente. En este capítulo nos centraremos en las dos últimas fases, que resultan estar íntimamente unidas entre sí, dando lugar a una serie de analizadores que tienen como objetivo elaborar una red bayesiana consistente. Las etapas que cubren estas dos fases son las siguientes:

- a) Extracción de las dependencias e independencias existentes entre las variables.- Esta fase consiste en la agrupación de cada uno de los efectos con las causas de las que depende.
- b) Actuación del Analizador Probabilístico que inicialice la Red Bayesiana.- En esta fase se procede a solicitar al experto las probabilidades a priori y las probabilidades condicionadas necesarias, puesto que todo nodo de la red, equivale a una matriz de probabilidades condicionadas. En la figura 7.3. se muestra gráficamente el funcionamiento de este analizador.
- c) Actuación del Analizador Espacial para crear una estructura de nodos y aristas donde representar las variables y las relaciones causales respectivamente.- Este analizador es el encargado de construir la red bayesiana con todas sus dependencias, con él elaboraremos una matriz de filas y columnas de manera imaginaria, sobre la cual situaremos cada uno de los marcos causales, en forma de nodos y aristas. Los nodos equivalen a los efectos y causas y las aristas a las relaciones de dependencia entre ellas.
- d) Actuación del Analizador de Consistencia que valide y verifique las probabilidades introducidas.- Este analizador comprueba que los datos probabilísticos introducidos por el usuario son correctos y que están dentro de los rangos adecuados.

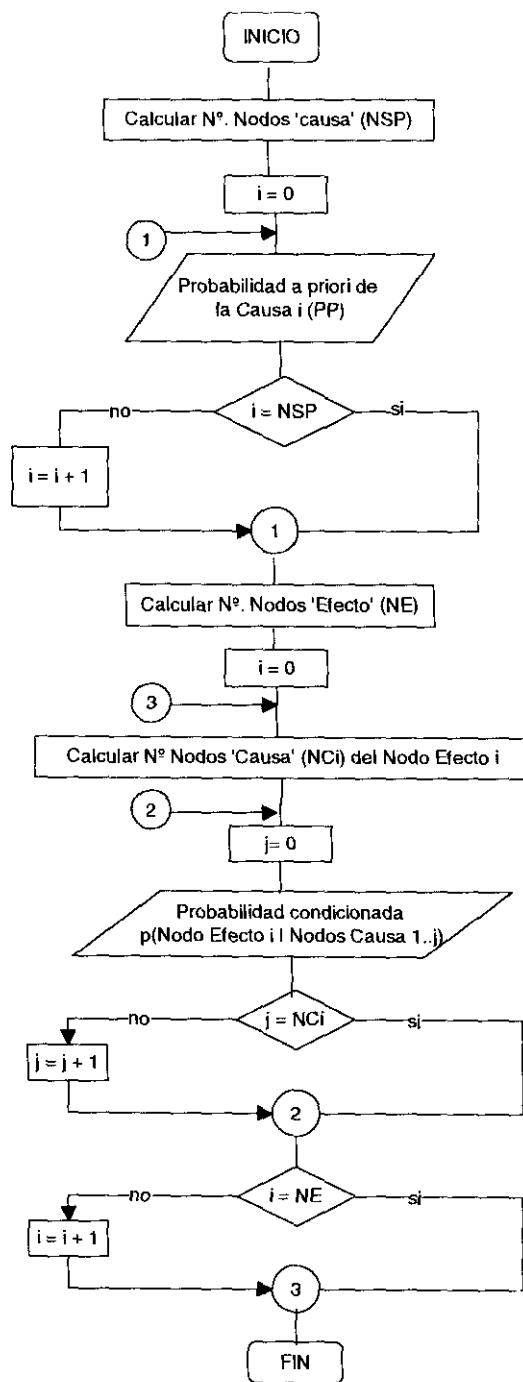


Figura 7.3. Funcionamiento del analizador probabilístico de redes bayesianas

# **Capítulo 8**

---

## **Subsistema de ejecución**

### **8.1. Presentación del conocimiento**

El módulo de presentación del conocimiento organiza el conocimiento con el fin de que el proceso de propagación sea ágil y proporcione tiempos aceptables de respuesta.

#### **8.1.1. Árboles de toma de decisión**

Los árboles de decisión constituyen una forma jerárquica de presentar el conocimiento almacenado en la base de conocimiento con el fin de facilitar la búsqueda y extracción del mismo. Fueron introducidos por Quinlan [144] y su estructura está formada por un árbol binario que consta de un conjunto de nudos y de ramas, y cuya creación responde al criterio de clasificación, rellenándose

los nodos de arriba a abajo según los que proporcionan una mayor exclusión de otros nodos y un direccionamiento mayor hacia una posible solución.

Las reglas de producción y los marcos de clasificación admiten este modelo de presentación como estructura básica para la propagación del conocimiento. Si la base de conocimiento está compuesta por reglas de producción, se recogen las líneas de antecedente o premisas de las reglas según se encuentren ordenadas en la misma regla, situándolas en forma descendente en el árbol, tal y como se indica en la figura 8.1. Si la base de conocimiento está basada en marcos de clasificación, el árbol de decisión organiza los nudos de cada uno de los marcos, de arriba a abajo, según el orden en que se presentan las ranuras en el marco, tal y como se muestra en la figura 8.2.

Los nudos de un árbol de toma de decisiones se emplean para representar las unidades y subunidades del conocimiento y las ramas unen los nudos entre sí y les proporcionan valores booleanos. Consideramos que en el módulo propuesto, los árboles de decisión están formados por tres tipos de nudos:

- **Nudo tipo ‘terna’.**- Representan los nudos tipo ‘atributo/operador/valor’, que se corresponden con las premisas en el caso de reglas de producción y con las ranuras en el caso de los marcos de clasificación.
- **Nudo tipo ‘objeto’.**- Representan los nudos tipo ‘objeto/operador/valor’, existentes tan sólo en el caso de las reglas de producción.
- **Nudo tipo ‘meta’.**- Representan los nudos tipo ‘hipótesis’, en caso de reglas de producción, o los nudos tipo ‘marco’ en caso de los marcos de clasificación.

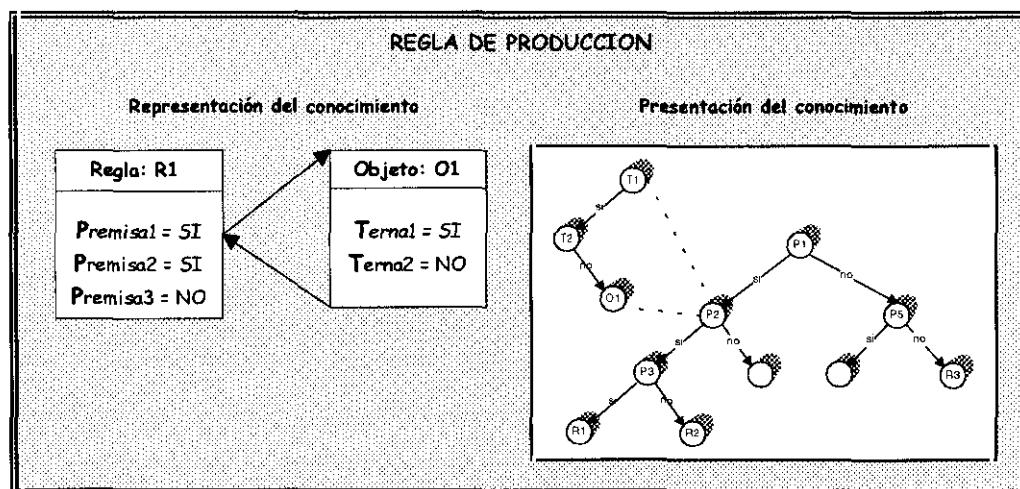


Figura 8.1. Árbol de decisión para una base de conocimiento basada en reglas de producción

Cuando se completa todo el antecedente de una regla, se desemboca en un nudo 'meta' que se traduce en una hipótesis y que a su vez puede relacionarse con otros nudos. Por otra parte cuando se completan todas las ranuras de un marco, se desemboca en un nudo 'meta' que se traduce en la propia meta del marco.

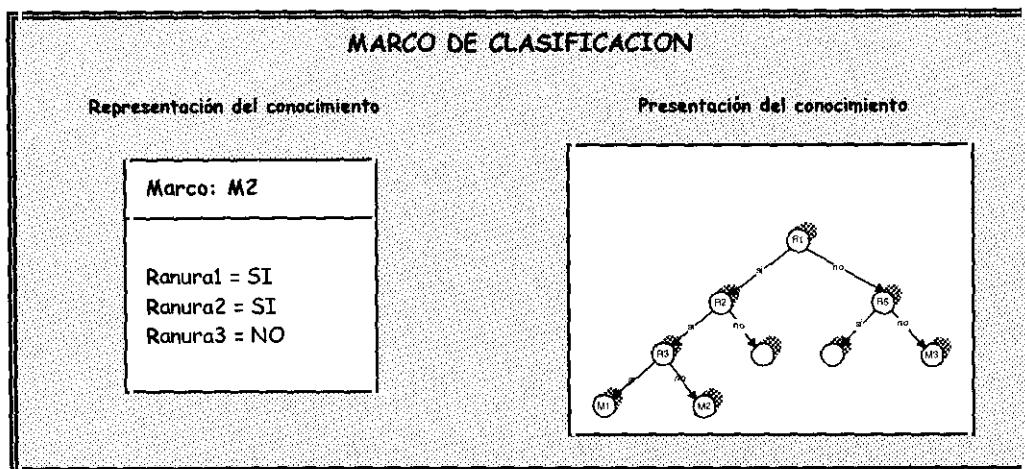


Figura 8.2. Árbol de decisión para una base de conocimiento basada en marcos de clasificación

La figura 8.1. muestra un ejemplo de cómo se presenta el conocimiento en forma de árbol de decisión cuando se dispone de una regla de producción, en la cual una de sus líneas de antecedente, en concreto la primera, es la afirmación de un objeto. La creación del árbol sigue el orden de las premisas, tal y como se encuentran en la regla, por lo tanto primero aparecen los nudos terna del objeto, y a continuación aparecen los nudos terna, con contenido de premisa, de la regla, para finalmente acabar una de las ramas del árbol con el nudo: R1, es decir, la regla 1.

La figura 8.2, al igual que la figura anterior, expone un ejemplo de un árbol de decisión para un marco de clasificación. En él, el orden que siguen los nudos es el mismo que las ranuras tienen en el marco correspondiente. Si seguimos la rama del marco: 'M2', vemos que a él se llega afirmando las ranuras 1 y 2 y negando la tercera, finalizando la rama del árbol en el nudo meta etiquetado como; 'M2'.

### 8.1.2. Redes de conceptos

Las redes de conceptos constituyen una forma de presentación del conocimiento en el momento previo a llevar a cabo una sesión de consulta en el subsistema de ejecución. Se denominan SMCPC

y están constituidas por una estructura multinivel-multinodo, de manera que cada nivel está formado por múltiples nodos que se relacionan, generalmente, con otros de un nivel contiguo, aunque existen casos en los que la red establece relaciones entre nodos de niveles no contiguos, y que se explicarán más adelante.

Las redes de conceptos pueden formarse, tanto a partir de bases de conocimiento basadas en reglas de producción, como en marcos de clasificación. La elección de las redes de conceptos como mecanismo de presentación del conocimiento, trae como consecuencia la aparición de dos tipos de estructuras: las proposiciones y los conocimientos. Las proposiciones unen las hipótesis o las metas de los marcos, mediante operadores propios del sistema o bien de nueva creación. Los conocimientos unen las proposiciones anteriormente creadas.

Una red de conceptos está formada por una serie de niveles, que se disponen en el espacio de abajo a arriba, desde el nivel más simple al más complejo, respectivamente. La creación de la red de conceptos pasa por la elaboración de diferentes etapas:

- a) Creación de los niveles genéricos de la red.- Durante esta etapa se trata de establecer qué tipos de unidades del conocimiento van a constituir los niveles de la red de conceptos. Esta etapa depende directamente del tipo de representación del conocimiento que se haya elegido. Para facilitar hemos convenido dar un valor numérico a cada una de las unidades del conocimiento, tal y como se muestra en la tabla 8.1.

Por otro lado el número de los niveles de la red va a variar según la representación elegida, así en el caso de las reglas de producción, la red de conceptos tiene cinco niveles y en el caso de los marcos, tiene cuatro.

Codificación de niveles	
0	Objeto
1	Atributo - Operador - Valor
2	Hipótesis
3	Marco
4	Proposición
5	Conocimiento

Tabla 8.1. Niveles genéricos de la red de conceptos

- b) Creación de los niveles específicos de la red.- Esta etapa consiste en ir rellenando cada uno de los niveles de la red de conceptos con la unidad de conocimiento adecuada. A continuación detallamos los componentes de cada uno de los niveles tanto para las reglas de producción como para los marcos de clasificación. En la figura 8.3 se muestran las redes de conceptos para reglas y para marcos.

### b.1) Reglas de producción

- 1er. Nivel.- Formado por las ternas: atributo-operador-valor, ya sea las constituyentes de las premisas o bien las constituyentes de los objetos.
- 2º. Nivel.- Formado por los objetos presentes en la base de conocimiento.
- 3er. Nivel.- Formado por las hipótesis de las reglas de producción.
- 4º. Nivel.- Formado por las proposiciones.
- 5º. Nivel.- Formado por los conocimientos.

### b.2) Marcos de clasificación

- 1er. Nivel.- Formado por las ranuras de los marcos.
- 2º. Nivel.- Formado por las metas de los marcos.
- 3er. Nivel.- Formado por las proposiciones.
- 4º. Nivel.- Formado por los conocimientos.

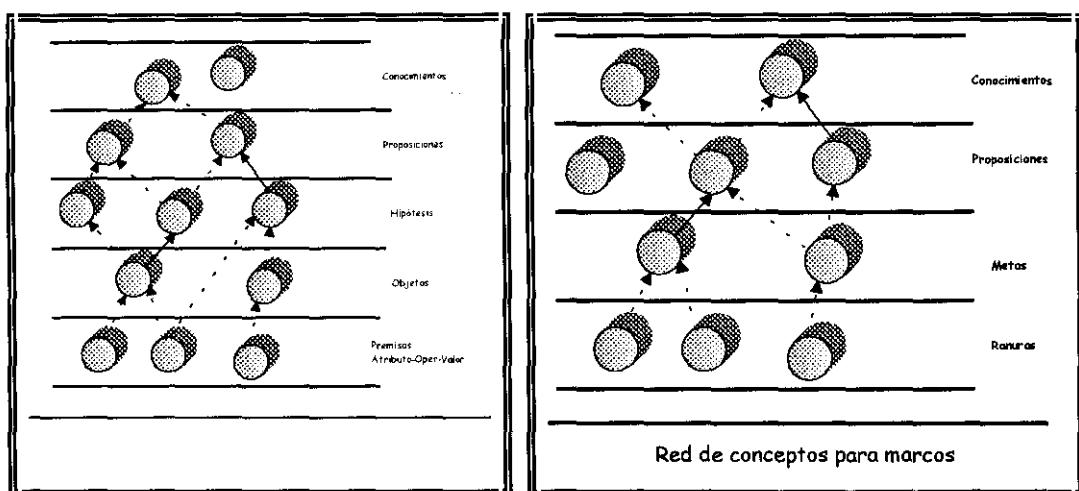


Figura 8.3. Niveles específicos de la red de conceptos

Como podemos observar, generalmente los nodos se forman por agregaciones de nodos de niveles anteriores, salvo en el caso particular de las reglas de producción, en que, al estar formadas éstas por la unión de ternas atributo-operador-valor con objetos en su antecedente, los nodos en algunas ocasiones no pasan el nivel contiguo sino que se saltan un nivel. Por otro lado, la red de

conceptos admite la existencia de nodos aislados, no unidos a ningún otro nodo. Estos nodos se refieren a aquellas evidencias que tiene el experto sobre algo, pero que de momento no puede atribuir a ningún conocimiento concreto.

Las redes de conceptos presentan una gran ventaja frente a los árboles por su capacidad de introducir conocimiento incompleto, pero que, en cualquier momento, puede convertirse en conocimiento completo cuando se une al ya existente.

## 8.2. Propagación del conocimiento

El módulo de propagación del conocimiento constituye el corazón de todo sistema basado en el conocimiento, puesto que es el encargado de llevar a cabo las consultas propiamente dichas.

### 8.2.1. Funcionamiento general

El funcionamiento general del módulo de propagación del conocimiento tiene dos partes: una de ellas común tanto a árboles como a redes de conceptos, y la otra particular para cada tipo de presentación. A continuación enumeramos y detallamos cada una de las etapas por las que atraviesa el módulo de propagación del conocimiento.

- a) Preparación del motor de inferencia
- b) Comienzo de la ejecución de una sesión
- c) Propagación del conocimiento

- a) Preparación del motor de inferencia.- Durante esta etapa se almacenan las posibles preguntas y metas, acordes con la representación de que disponga la base de conocimiento, con la que va a trabajar el motor de inferencia. Según el tipo de representación nos encontramos:

- a.1) Reglas de producción
  - Extracción de las hipótesis de cada regla y almacenamiento en tabla.
  - Extracción de las premisas de cada regla, que se convertirán en futuras preguntas del sistema, y almacenamiento en tabla.

- Extracción de los objetos y almacenamiento en tabla.
- Extracción de las ternas ‘atributo-operador-valor’ de cada uno de los objetos y almacenamiento en tabla.

a.2) Marcos de clasificación

- Extracción de los marcos y almacenamiento en tabla.
- Extracción de las ranuras de cada marco y almacenamiento en tabla.

a.3) Marcos causales

- Extracción de los efectos de los marcos causales y almacenamiento en tabla
- Extracción de las causas de cada marco causal y almacenamiento en tabla.

De esta forma disponemos de todas las posibles metas y preguntas almacenadas en una tabla. Cada uno de estos elementos almacenados, tendrá un estado de chequeo que a lo largo del funcionamiento del motor de inferencia adquirirá diferentes valores que detallamos a continuación. Inicialmente, preguntas y metas se encuentran con estado: ‘No evaluado’. En la tabla 8.2. ilustramos los estados de chequeo que hemos considerado.

Estados de chequeo	
0	No evaluado
1	En Evaluación
2	Afirmado
3	Pendiente de evaluar
4	Evaluación Automática
5	Negado

Tabla 8.2. Estados de chequeo del motor de inferencia

- b) Comienzo de la ejecución de una sesión.- Este punto es crucial para que el motor de inferencia empiece a funcionar. Para ello es necesario saber si el usuario dispone o no de información para alimentar el sistema. En este último caso debe ser el propio sistema el que mediante prioridades si las tuviera, o de manera aleatoria, siguiendo reglas propuestas por el autor, dará comienzo a la sesión de ejecución propiamente dicha. A continuación detallamos cada uno de estos aspectos.

b.1) Existen datos iniciales.- En el caso de que se disponga de algún tipo de conocimiento sobre el área a tratar, el sistema proporcionará al usuario las posibles preguntas y metas existentes en el sistema, según el tipo de representación elegida:

- Reglas de producción.- El sistema extrae de las tablas rellenas en la etapa anterior, tanto las hipótesis, los objetos y las ternas atributo-operador-valor, y las muestra por pantalla, para que el usuario escoja los elementos sobre los que tiene información.
- Marcos de clasificación.- El sistema extrae igualmente, los marcos y las ranuras, mostrándolas por pantalla para que el usuario elija igualmente lo que conoce.
- Marcos causales.- El sistema extrae las causas y los efectos, de la misma forma que en los apartados anteriores.

b.2) No existen datos iniciales.- Si el usuario no tiene conocimientos sobre el área a tratar, el sistema tiene dos formas de actuar, según el tipo de unidad de conocimiento:

- Reglas de producción.- Pueden ocurrir dos casos:
  - Si las reglas de producción tienen prioridades asociadas, el sistema ordena las reglas por orden de prioridad de mayor a menor, y comienza a preguntar por las de mayor prioridad.
  - En el caso de que no tengan prioridades, se busca el atributo más frecuente, y se ordenan las reglas de forma descendente, según la frecuencia del mismo en las reglas.
- Marcos de clasificación.- Los marcos se clasifican en orden a la frecuencia de sus atributos, empezando por aquellos que tengan en sus ranuras el atributo más frecuente.

c) Propagación del conocimiento.- El proceso de propagación consiste en conseguir llegar a una conclusión empleando el menor número de preguntas posible. De esta forma se han creado una serie de reglas lógicas de funcionamiento para cumplir este objetivo. A continuación se detalla dicha forma de funcionamiento.

- c.1) Si no existen preguntas pendientes de evaluar, ha de continuarse con la regla o marco de clasificación o causal que tenga menor número de premisas, ranuras o causas respectivamente, pendientes de evaluar.
- c.2) Si no se ha llegado a cumplir un objetivo (hipótesis, marco de clasificación o marco causal), debe continuarse preguntando el objetivo que esté activo.

c.3) Si se ha cumplido un objetivo, se debe proporcionar éste al usuario y comprobar si el motor de inferencia puede continuar funcionando y cubrir otros posibles objetivos. En el caso de las reglas, si se ha cubierto una hipótesis, inmediatamente se desencadena el consecuente asociado.

La propagación del conocimiento viene acompañada del funcionamiento del visor de trazas, que va proporcionando información de lo que va ocurriendo en el motor de inferencia.

### 8.2.2. Árboles de toma de decisión

La propagación del conocimiento basada en árboles de toma de decisiones se puede realizar cuando la base de conocimiento está constituida ya sea por reglas de producción o por marcos de clasificación. En el caso del patrón de arquitectura PARGEN, se propone la creación de árboles dicotómicos multinivel, que toman diferente estructura según se trate de reglas o de marcos. En el caso de las reglas de producción se escoge como presentación los árboles multiplano-multinivel, y en el caso de los marcos se trata de árboles multinivel tan sólo. A continuación comentamos qué queremos decir con cada uno de estos tipos de presentación.

- **Árboles dicotómicos.**- Se basan en que las premisas de las reglas o bien las ranuras de los marcos, se cumplen o no se cumplen, por lo tanto, de cada nodo del árbol, que servirá para representar las premisas y las ranuras, saldrán siempre dos ramas de tipo booleano.
- **Multiplano.**- Los árboles son multiplano porque en el caso de las reglas de producción, al admitir el operador 'O' como nexo de unión entre las premisas, se formarán varios árboles superpuestos de manera imaginaria, que completan al final de cada rama, una misma hipótesis. En el caso de los marcos, los árboles no tienen esta particularidad.
- **Multinivel.**- Decimos que los árboles son multinivel, tanto para las reglas como para los marcos, porque todo árbol va a tener varios niveles hasta llegar a la hipótesis de la regla o a la meta del marco.

De este modo, los árboles de decisión en el caso de las reglas de producción, están formados por tres dimensiones y en el caso de los marcos, por dos dimensiones, tal y como se ilustra en la figura 8.4. Así observamos que en las reglas de producción los planos paralelos se unen mediante el operador: 'O', y los niveles mediante el operador lógico: 'Y'. En los marcos de clasificación existe solamente un plano. La dimensión horizontal del sistema viene dada por la propia composición de la base de conocimiento.

En las figuras 8.5. y 8.6. se pone de manifiesto esta estructura de manera más específica. En la figura 8.5 podemos observar la estructura interna de cada nivel del árbol de decisión, según la cual, los nudos están organizados en niveles hasta llegar a los nudos que ya no tienen ninguno por debajo. Estos

nudos equivalentes a las metas propiamente dichas, y, por tanto, dan por terminada la sesión de ejecución cuando se llega a ellos.

La figura 8.6. proporciona información sobre la estructura interna del árbol de decisión para las reglas de producción. Aparecen dos planos en la estructura, puesto que existen varias reglas que comparten la misma hipótesis, los niveles, al igual que en el caso anterior, se crean cuando existen operadores 'Y' entre las premisas del antecedente.

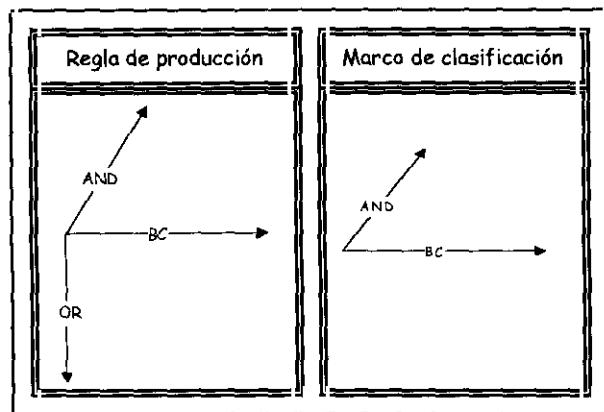


Figura 8.4. Estructura interna de los árboles de decisión

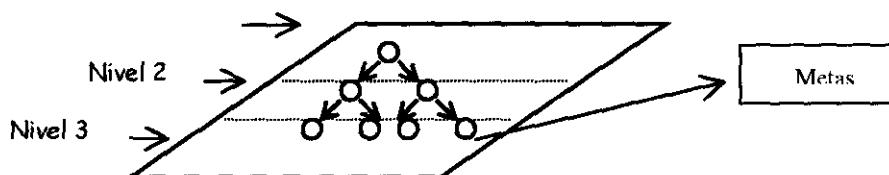


Figura 8.5. Árbol de decisión multinivel

De esta forma, una vez elegidos los árboles de decisión para presentar el conocimiento, se extraen las ranuras del marco activo o las premisas de la regla activa, para dar lugar a la construcción del árbol. Denominaremos activos a aquellos elementos pendientes de evaluar en el motor de inferencia.

La construcción de los árboles, en el caso de los marcos de clasificación, dará lugar a un único árbol de decisión, donde los nodos son cada una de las ranuras de un marco concreto y el nodo final de una rama concreta, es la meta del marco.

En el caso de las reglas de producción, es un poco más complejo, al tener que recurrir a las diferentes subreglas en que se descompone la regla activa, formando un árbol por cada una de ellas y

situándolos en el espacio, en planos paralelos entre sí. De esta forma a medida que se suceda la consulta, se puede producir el salto de un árbol a otro, según se vayan cumpliendo unas ramas u otras. Al final se completará una rama entera y se comprobará si la hipótesis a la que se ha llegado es afirmativa o negativa.

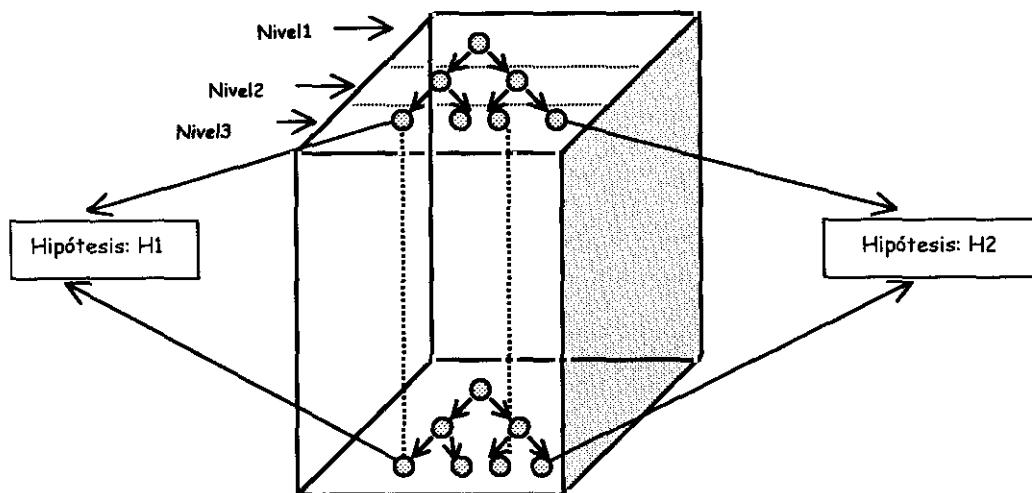


Figura 8.6. Árbol de decisión multiplano-multinivel

### 8.2.3. Redes de conceptos

La propagación del conocimiento empleando como forma de presentación del conocimiento las redes de conceptos. Haremos un breve resumen de su funcionamiento.

Las redes de conceptos se aplican tanto a reglas de producción como a marcos de clasificación y consisten en la creación de redes multinivel-multinodo, donde cada nivel está etiquetado por un tipo de unidad de conocimiento. En el caso de las reglas de producción, existe un nivel más que en el caso de los marcos de clasificación. En las figuras 8.7 y 8.8 se observa cómo se produce el proceso de propagación en las redes de conceptos basadas en marcos y en reglas, respectivamente.

La propagación comienza a partir de una regla activa o un marco activo, en cuyo caso se encienden todos los nodos pendientes de evaluar y comienzan a preguntarse. Según se van cumpliendo nodos, se van actualizando sus estados de chequeo, y, además, después de cada respuesta del usuario, se comprueba la evaluación automática, es decir, si existen posibles elementos que con lo contestado hasta el momento se pueda chequear. Una vez que el nivel de las hipótesis o el de las metas, tiene más de un nodo evaluado, se procede a observar si se puede deducir alguna proposición, y algún conocimiento como consecuencia.

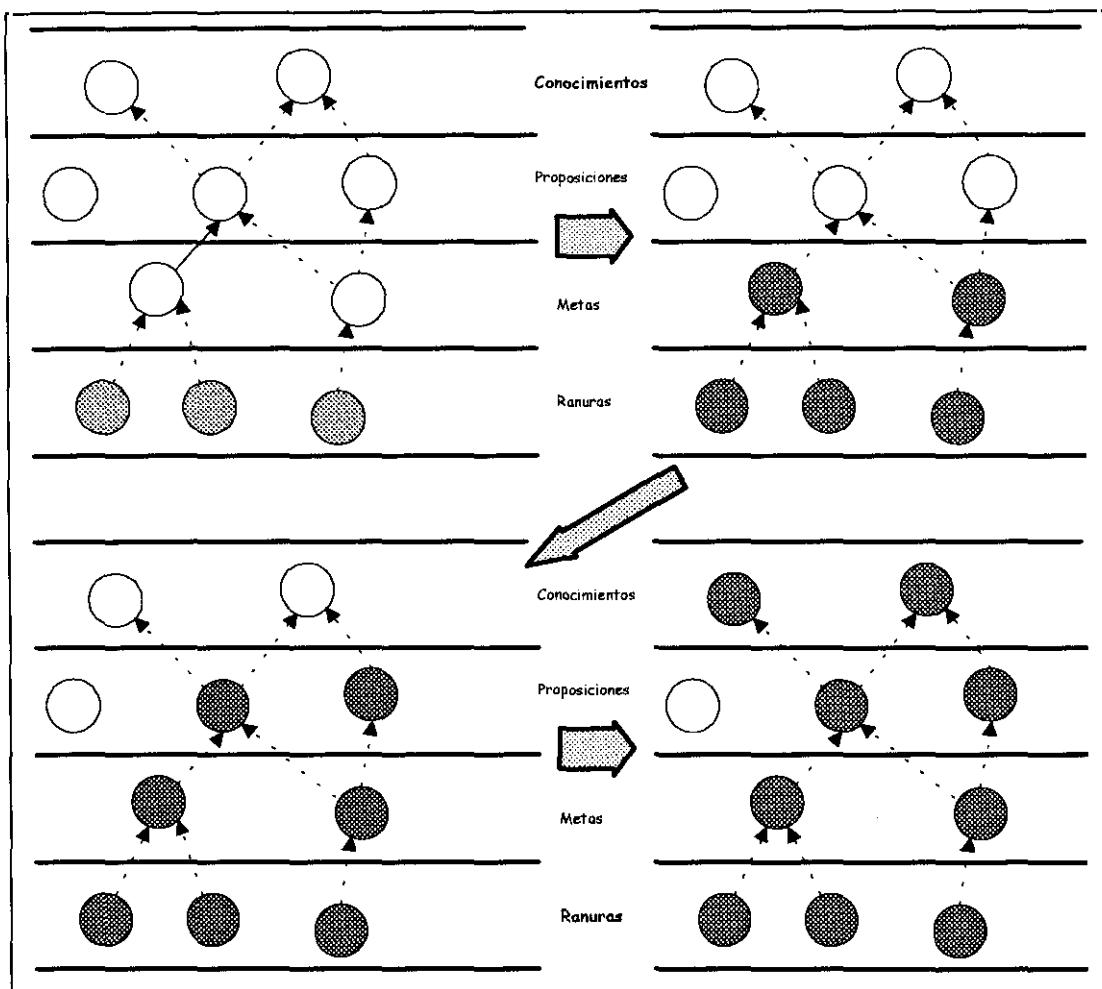


Figura 8.7. Propagación en una red de conceptos basada en marcos

En estas figuras hemos considerado los nodos de color blanco como pendientes de preguntar al usuario, los nodos de color gris claro como los que se están preguntando en un momento dado, los nodos gris oscuro a los nodos que ya han sido contestados directamente por el usuario o bien que se disparan automáticamente en cuanto ya se consideran afirmativos en la red por las respuestas previas del usuario y los nodos negros cuando son negados por el usuario.

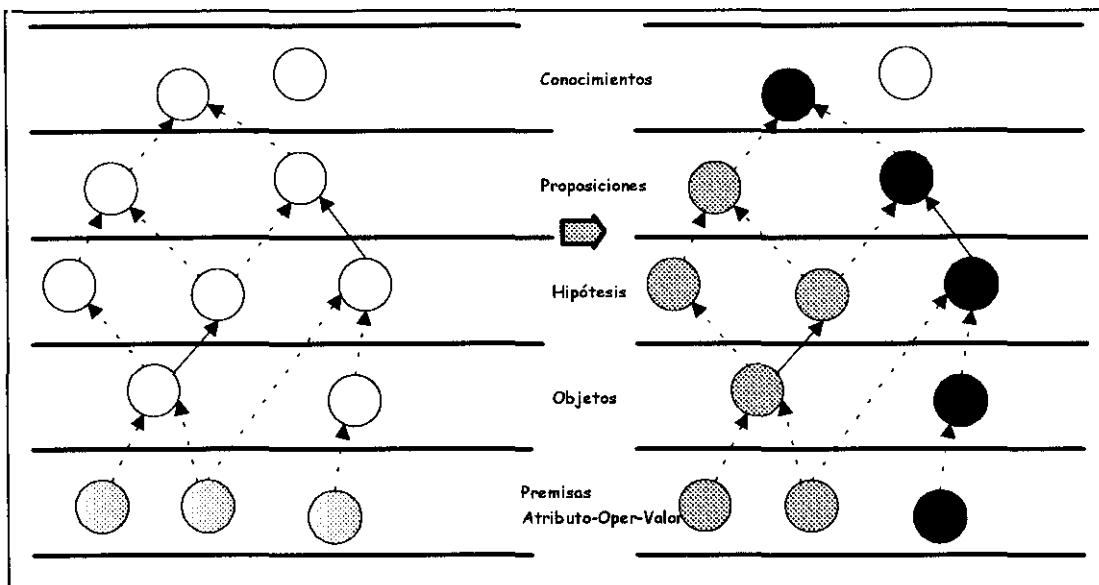


Figura 8.8. Propagación en una red de conceptos basada en reglas de producción

## 8.3. Propagación de la Incertidumbre

### 8.3.1. Probabilidad subjetiva

El proceso de propagación de la probabilidad subjetiva se desencadena una vez que se inicia la propagación del conocimiento y el motor de inferencia está preparado para empezar a interrogar al usuario. Este método de propagación está asociado a las reglas de producción. La elaboración de las medidas de suficiencia y necesidad, se basa en las probabilidades condicionadas, y el radio probabilístico se basa en las probabilidades a priori, tal y como se muestra en la figura 8.9.

En el patrón de arquitectura propuesto, los valores de las medidas de suficiencia y necesidad se preguntan directamente al usuario, a la vez que se le pregunta cada premisa, de forma que a partir de ellas y de las probabilidades a priori de cada hipótesis y de los objetos, se van propagando las probabilidades. En la figura 8.10. se muestra el funcionamiento genérico de este método de propagación. Para ello se siguen una serie de pasos que detallamos a continuación, una vez que se dispone de los datos probabilísticos completos: probabilidades a priori y medidas de suficiencia y necesidad.

- a) Calcular las razones de probabilidad previa o radios probabilísticos de hipótesis y objetos:  $\Phi(H)$ .
- b) Comprobar si se trata de la primera premisa de la regla. Si lo fuera, se hace una asignación de la razón de probabilidad previa a la razón de probabilidad de la premisa anterior:

$$\Phi(i, j - 1) = \Phi(i)$$

- c) Calcular la razón de probabilidad posterior, aplicando el producto a la razón de probabilidad obtenida en el apartado anterior y a la medida de suficiencia:

$$\Phi(i, j) = \Phi(i, j - 1) * MS(i, j)$$

- d) Calcular las probabilidades condicionadas de la premisa actual respecto a su hipótesis.

$$P(i | j) = \frac{\Phi(i, j)}{1 + \Phi(i, j)}$$

De esta forma se va obteniendo una propagación de las probabilidades a medida que transcurre el proceso de inferencia.

### 8.3.2. Factores de certeza

La propagación de los factores de certeza sigue un algoritmo sencillo que consta de varios pasos, que se muestran en la figura 8.11. Estos pasos se pueden estructurar en los siguientes:

- a) Calcular el número de premisas de la regla que se encuentre activa en ese momento.
- b) Preguntar el factor de certeza de la premisa que se encuentre activa, es decir, con el estado ‘pendiente de evaluar’.
- c) Evaluar el factor de certeza introducido y si tuviera un valor no significativo ( $CF < 0.2$ ), se le asigna automáticamente el valor 0.
- d) Si no fuera la primera premisa de la regla activa, analizar qué operador lógico le une a la premisa que le precede, y según sea ‘Y’ u ‘O’, llevar a cabo unas operaciones u otras.

- e) Calcular el factor de certeza final de la regla una vez que se han evaluado todas las premisas de la regla activa.

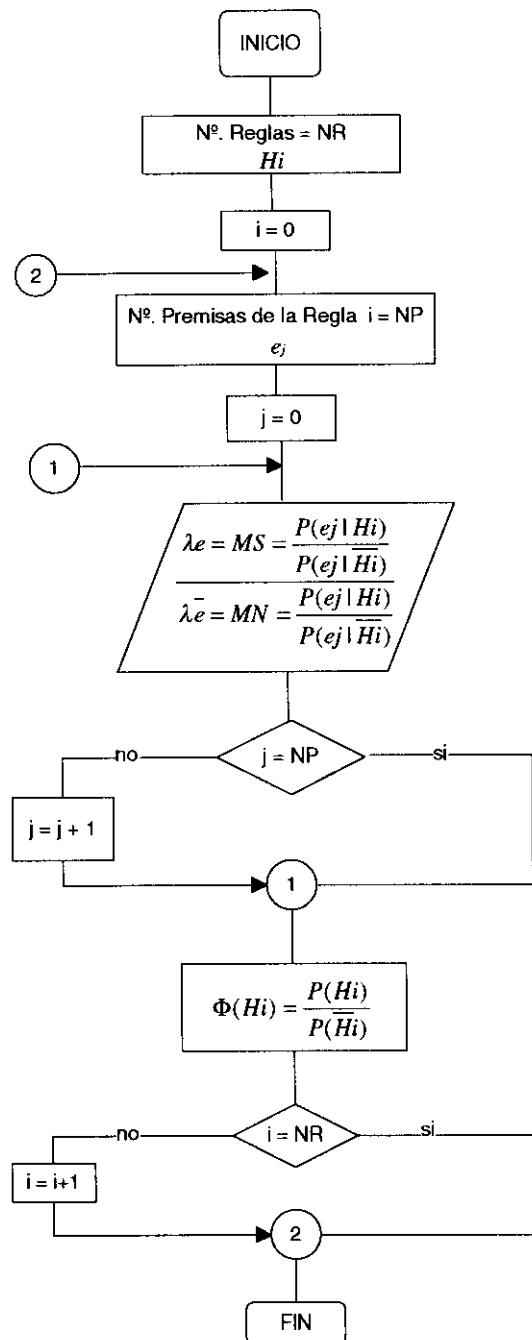


Figura 8.9. Elaboración de las medidas de suficiencia y necesidad

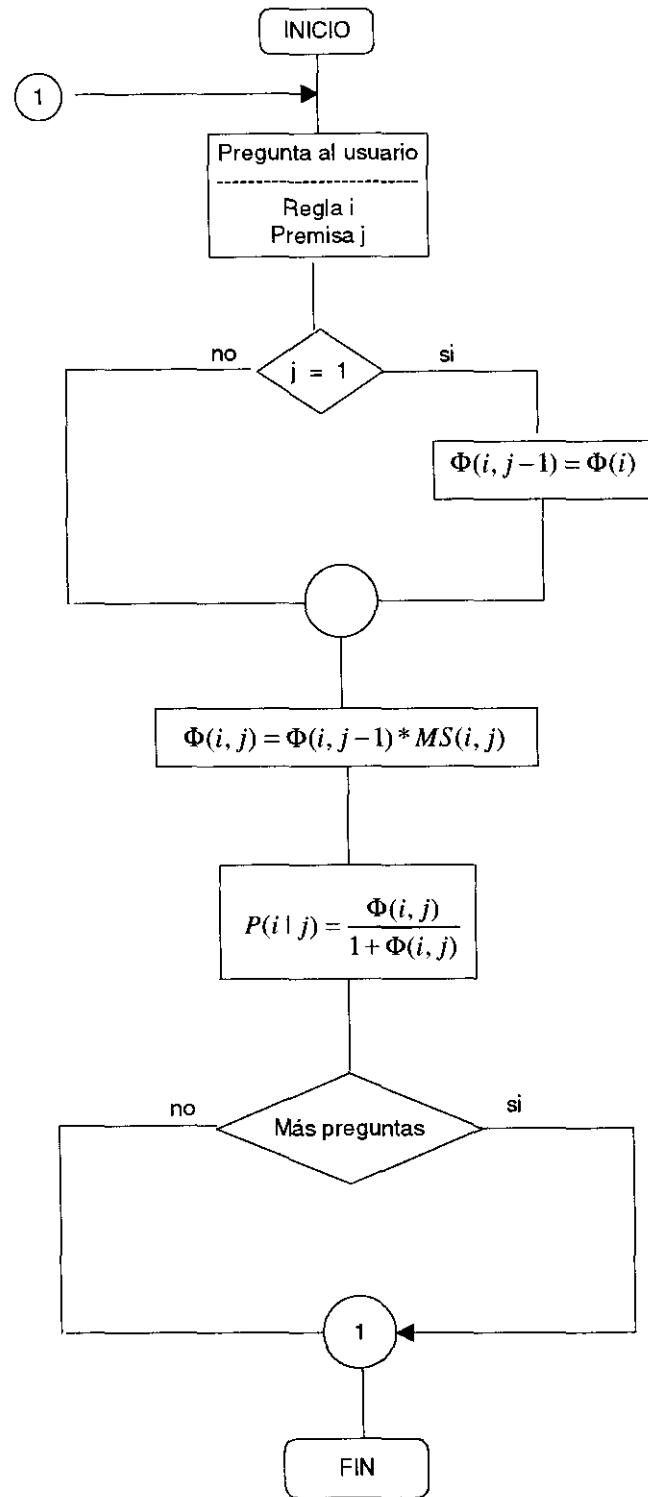


Figura 8.10. Propagación de la probabilidad subjetiva

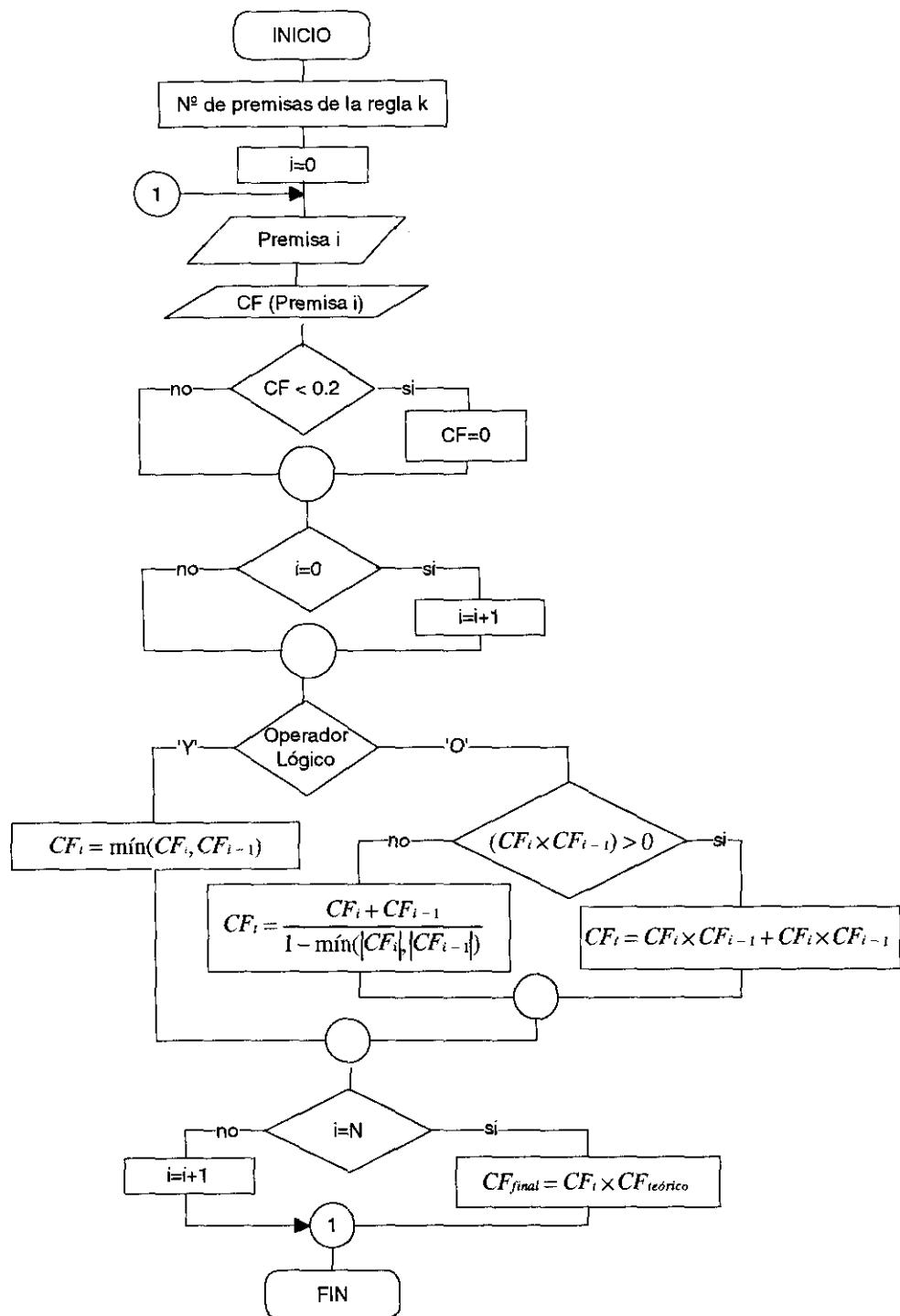


Figura 8.11. Propagación de los factores de certeza

### 8.3.3. Redes bayesianas

El patrón de arquitectura propuesto presenta en su módulo de propagación de la incertidumbre, un tipo particular de representación del conocimiento: los marcos causales. Estos ofrecen la particularidad de transformarse directamente en una red bayesiana en el momento de la puesta en marcha de la sesión de ejecución. La propagación de la red bayesiana se basa en el paso de mensajes de unos nodos a otros dentro de la red, de forma que se van calculando las probabilidades asociadas a cada nodo. En este caso particular hemos empleado como algoritmo de propagación, uno asociado a poliárboles, [98, 133], cuya implementación se puede ver en [73].

El funcionamiento correcto de este algoritmo depende de dos circunstancias:

- a) Consistencia estructural de la base de conocimiento. - El módulo de validación y verificación, mediante su analizador de coherencia total, se encarga de evitar que existan ciclos en la base de conocimiento formada por los marcos causales. De esta forma la red bayesiana responderá a una estructura de políárbol, de forma que dos nodos se unirán entre sí mediante un sólo camino, pudiéndose aplicar de esta forma algoritmos de propagación de poliárboles.
- b) Consistencia numérica de la base de conocimiento. - La propagación correcta de las probabilidades depende directamente de que los datos iniciales, de los que parte la red bayesiana y que han sido introducidos por el usuario, estén dentro de los rangos posibles.

El funcionamiento del algoritmo de propagación propiamente dicho pasa por varias fases:

- a) Petición al usuario de todas las probabilidades: probabilidades a priori del efecto del marco y probabilidades condicionadas de cada una de las causas.
- b) Comprobación de la consistencia de las probabilidades introducidas.
- c) Creación estructural de la red bayesiana, que consta de los siguientes pasos:
  - c.1) Creación de nodos y aristas a partir de los marcos causales, teniendo en cuenta que los nodos representan los marcos y las causas y aristas representan las relaciones: '*causa – marco*'.
  - c.2) Optimización de dichos nodos y aristas, de forma que si dos nodos etiquetados de forma diferente, realmente responden a la misma causa, pasan a ser el mismo nodo.
  - c.3) Estudio de las relaciones causales existentes entre los diferentes nodos con el fin de distribuirlos en niveles imaginarios que van de arriba a abajo como si de una relación jerárquica se tratara. Los nodos que son causa de otros, se denominarán a partir de ahora, nodos 'padre' y los nodos efecto de otras causas se denominarán nodos 'hijo'.

c.4) Estudio de las causas que intervienen en cada uno de los marcos, con el fin de distribuir en columnas la red bayesiana. Como ejemplo podemos decir que una causa que intervenga en dos marcos, debe estar en columnas contiguas, para evitar cruces de relaciones en la estructura de la red.

d) Propagación de las probabilidades, que consiste en la siguiente secuencia de pasos:

- d.1) Búsqueda en la red bayesiana de los nodos que no tienen padres y asignación de su probabilidad. Esta probabilidad es en realidad la probabilidad a priori introducida por el usuario inicialmente.
- d.2) Búsqueda en la red bayesiana de los nodos que no tienen hijos y asignación de su probabilidad inicial. En principio le asignaremos el valor uno.
- d.3) Búsqueda sucesiva de nodos que tienen padres e hijos en número creciente empezando en 0 y 1 respectivamente, alternando con los que tienen hijos y padres en número 1 y 0 respectivamente. El algoritmo consiste en ir calculando de manera iterativa, las probabilidades debidas a los padres, a partir de ahora etiquetadas como  $\pi$  y las probabilidades debidas a los hijos, etiquetadas como  $\lambda$ . Al final dispondremos de una estructura en poliárbol donde todos los nodos tienen dos probabilidades, una de ellas se basa en la contribución de todos los nodos que pertenecen a un nivel inferior, es decir, a los nodos ‘padre’, y la otra debida a la contribución de todos los nodos que pertenecen a un nivel superior respecto a Él, es decir, a los nodos ‘hijo’.

A continuación, en la figura 8.12. exponemos a modo de esquema, el funcionamiento del algoritmo implementado para propagar las probabilidades por la red bayesiana.

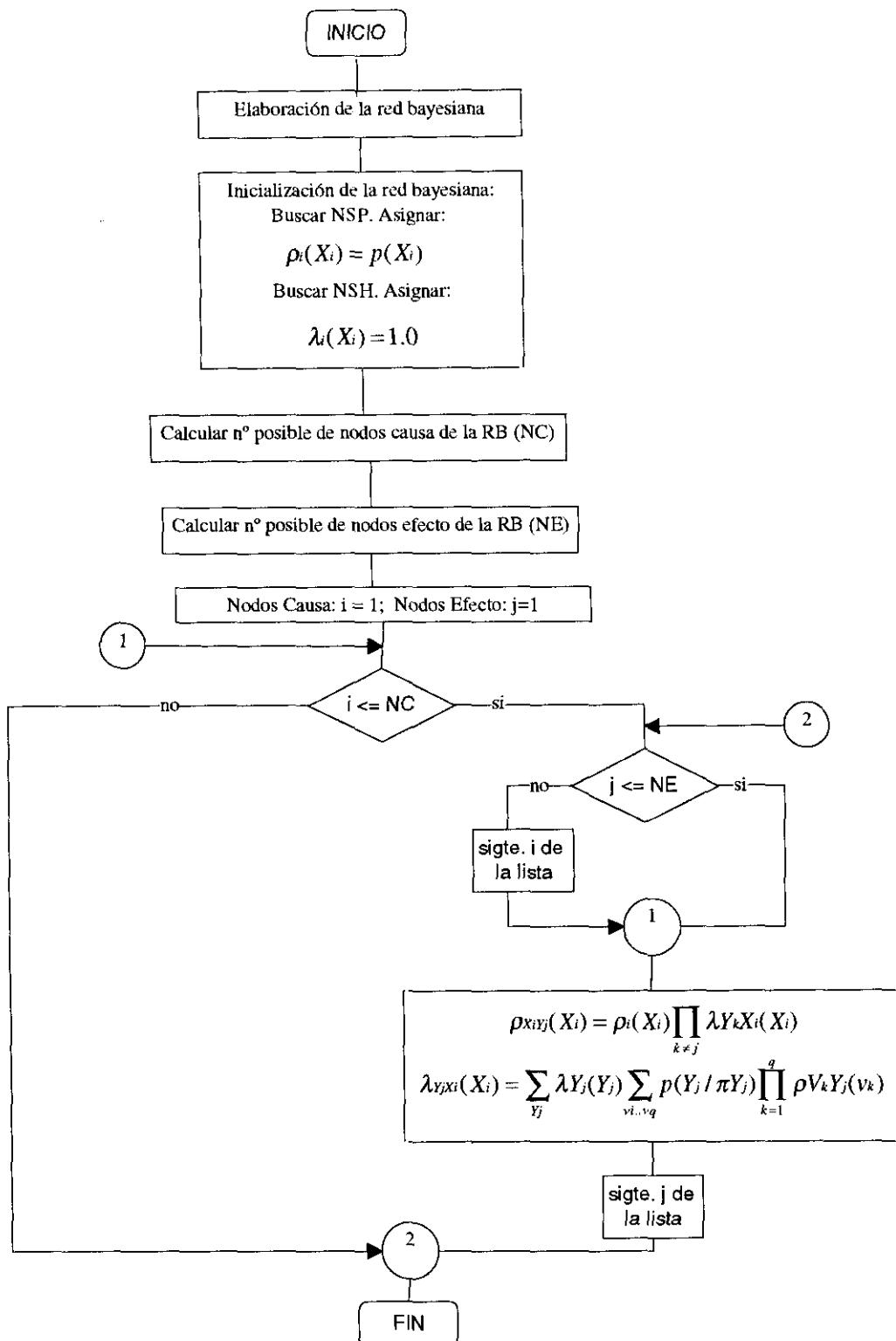
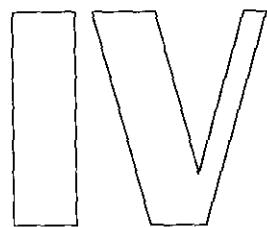


Figura 8.12. Propagación de la red bayesiana



# **DESARROLLO INFORMÁTICO**

**INTEGRACIÓN DEL LENGUAJE DE PATRONES  
CON EL PATRÓN DE ARQUITECTURA  
IMPLEMENTACIÓN**

# **Capítulo 9**

---

## **Integración del lenguaje de patrones con el patrón de arquitectura**

### **9.1. Captura de requisitos**

Todo desarrollo informático requiere una primera fase de captura de requisitos, entendiendo por tal el conjunto de necesidades que debe cumplir un sistema determinado y que facilita el usuario que ha propuesto la formalización del problema real. Puesto que en el anexo se ha desarrollado un caso práctico para explicar que los patrones de diseño propuestos desembocan en un desarrollo concreto, nos ha parecido conveniente proponer una forma sencilla de capturar las necesidades que ha de cumplir el sistema informático a elaborar. Esta etapa está totalmente orientada al patrón de arquitectura PARGEN, viene a ser una forma de ordenar las peticiones del usuario; se lleva a cabo mediante una serie de reuniones entre usuario y desarrollador. En un principio se tendrá un conocimiento puramente descriptivo del problema, y a medida que se vayan sucediendo las reuniones,

se irá ascendiendo en complejidad. Entendemos que los objetivos de la captura de requisitos pueden agruparse en los siguientes:

- Extraer las **entidades** que va a manejar el sistema
- Extraer la **funcionalidad** del sistema.
- Extraer las **relaciones** entre las diferentes entidades para llevar a cabo dicha funcionalidad.

Puesto que consideramos un requisito como una necesidad que debe cumplir un sistema, debe expresarse atendiendo a un formato concreto, de la siguiente forma:

- Mediante una frase corta, redactada en voz activa (no pasiva), con sus componentes atendiendo al orden estándar de una frase: sujeto, verbo y complemento.
- Mediante una frase subordinada, de tipo condicional.

Proponemos en esta etapa la formalización de una serie de fichas que presentan un formato predefinido, con el fin facilitar por un lado la extracción de las peticiones formuladas por el usuario, y por otro, la subsiguiente elaboración del análisis y diseño. Con estas fichas, es prácticamente inmediata la transposición de las necesidades del sistema a las entidades conceptuales y funcionales de las siguientes etapas de desarrollo. Las fichas que se proponen requieren de un cierto trabajo anterior, por parte del desarrollador, para llenarlas correctamente. Responden a tres tipos:

- a) **Fichas Descriptivas**.- Plasman la descripción genérica del sistema.
- b) **Fichas Conceptuales**.- Concretan las entidades del sistema.
- c) **Fichas Funcionales**.- Detallan la funcionalidad del sistema.
- d) **Fichas Visuales**.- Muestran posibles diseños de pantallas para la introducción de la información.

### 9.1.1. Fichas Descriptivas

Las fichas descriptivas se emplean para establecer una primera aproximación del problema real, partiendo del patrón de arquitectura propuesto resulta relativamente sencillo llenarlas, puesto que el patrón ya proporciona la subdivisión en subsistemas y módulos. La formalización se expone en forma

de varias fichas descriptivas atendiendo al ámbito en que nos encontremos: aplicación, subsistema o módulo, de la forma en que se expone en la figura 9.1.

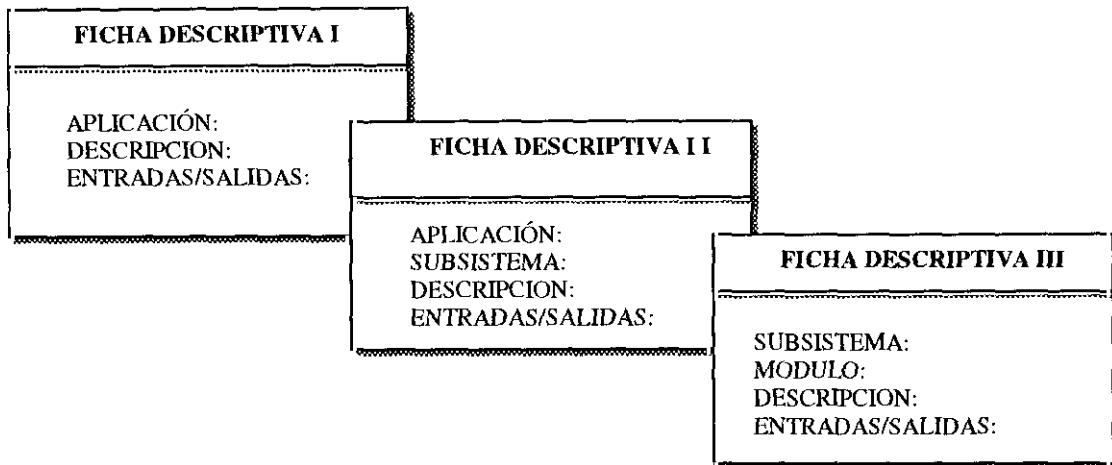


Figura 9.1. Esquemas de fichas descriptivas

### 9.1.2. Fichas Conceptuales

Las fichas conceptuales documentan las entidades conceptuales del sistema: su descripción, partes de que consta, con qué otras entidades están relacionadas y el papel que juegan en el sistema. Si se diera una acotación de tipos para una entidad concreta, éste es el momento idóneo para plasmarla. Las fichas conceptuales están formadas por las siguientes fases, que a continuación detallaremos más profundamente:

- Descomposición conceptual,
  - Descripción conceptual y
  - Descripción de tipos
- 
- La descomposición conceptual consiste en la extracción de todas las entidades que tienen una participación activa en el problema a formalizar. Resulta muy difícil conocer el número total de entidades desde las primeras tomas de contacto con el usuario, por tanto será necesario elaborar las fichas conceptuales en diferentes momentos del proceso de extracción del conocimiento del usuario. La descomposición conceptual se lleva a cabo mediante la enumeración de los conceptos a modo de lista en una ficha etiquetada como: 'Ficha Conceptual I' y cuyo esquema responde al presentado en la figura 9.2.

- b) La descripción conceptual consiste en definir cada una de las entidades extraídas durante la fase de la descomposición conceptual. Esta definición habrá de proporcionar información acerca de lo que significa dicha entidad en el sistema, sus relaciones con otras entidades así como las propiedades que la determinan. La descripción conceptual se plasma por medio de frases cortas y se lleva a cabo mediante la elaboración de una ficha etiquetada como: 'Ficha Conceptual II' y cuyo esquema responde al mostrado en la figura 9.2.

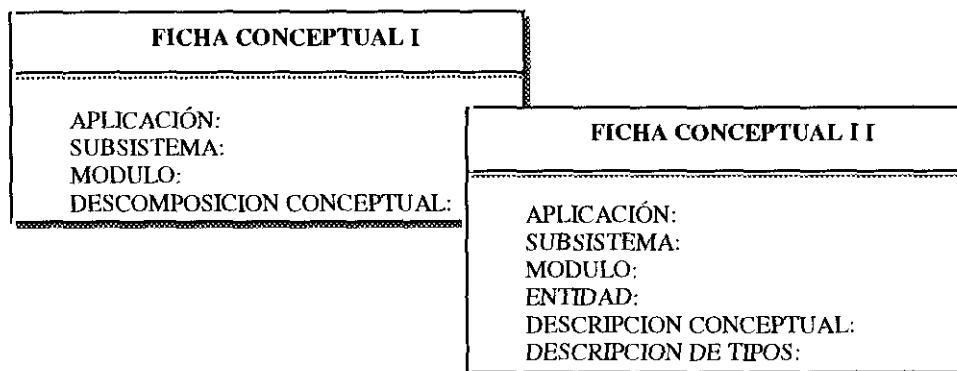


Figura 9.2. Esquema de fichas conceptuales

- c) La descripción de tipos consiste en concretar los valores determinados que puede tomar una determinada entidad. La ficha conceptual II, empleada en el apartado anterior, se usa igualmente para describir los posibles tipos que pueden tomar las entidades.

### 9.1.3. Fichas Funcionales

Las fichas funcionales se encargan de documentar las entidades funcionales del sistema, tanto desde el punto de vista descriptivo, como desde el punto de vista de la descomposición funcional. Para ello se proponen dos modelos de ficha que se ilustran en la figura 9.3. y que se detallan a continuación:

- a) Descripción funcional y
  - b) Descomposición funcional
- a) La descripción funcional consiste en definir el funcionamiento genérico que va a tener el sistema, subsistema o módulo que se trate. En esta fase van a ser esenciales los verbos que se empleen,

puesto que de eso dependerá en muchos casos que la aplicación tenga una determinada funcionalidad u otra. La descripción funcional se lleva a cabo mediante la elaboración de frases cortas, normalmente transcritas directamente de las conversaciones con el usuario. El sujeto de estas frases suele ser el propio sistema, subsistema o módulo.

- b) La descomposición funcional consiste en dilucidar todas y cada una de las tareas funcionales de las entidades extraídas mediante las fichas conceptuales. De esta forma, la descomposición funcional se representa mediante frases cortas que tendrán como sujeto dichas entidades, plasmando las operaciones a realizar con ellas, las condiciones existentes para realizar dichas operaciones, caso de haberlas, y las interrelaciones con otras entidades de la aplicación.

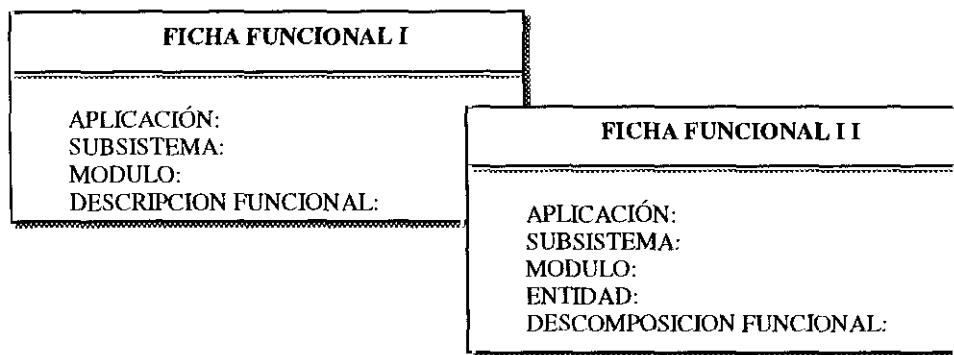


Figura 9.3. Esquema de fichas funcionales

#### 9.1.4. Fichas Visuales

Las fichas visuales documentan todo lo relacionado con la interfaz gráfica de usuario. De esta forma se hacen posibles esquemas de pantallas que el usuario deseé para llevar a cabo las operaciones concretas a través de la aplicación. Este tipo de fichas responde a dos tipos que se citan a continuación:

- a) Ficha con diseño visual de pantalla y
  - b) Ficha con la navegación por pantallas
- 
- a) Las fichas con el diseño visual de pantalla proporcionan una idea del aspecto que pueden tener las pantallas de la aplicación a construir. Este diseño lo lleva a cabo el usuario en colaboración con el desarrollador.

- b) Las fichas con la navegación por pantallas detallan gráficamente las conexiones que han de existir entre pantallas con el fin de que se comuniquen unas con otras y completen satisfactoriamente la funcionalidad requerida para la aplicación. Este tipo de fichas las realiza el desarrollador en colaboración con el usuario y son pantallas dibujadas a mano, interrelacionadas entre sí mediante flechas que salen de botones u otro tipo de controles que posea la pantalla.

Ambas fichas deben ser sencillas de elaborar y fáciles de interpretar.

### 9.1.5. Extracción de elementos

Durante la etapa de captura de requisitos, el desarrollador extrae en forma de frases cortas toda la información que le facilita el usuario. A partir de dichas frases cortas, es necesario extraer todo el conjunto de entidades conceptuales y funcionales que se va a necesitar durante el desarrollo de la aplicación. Se proponen una serie de reglas sintácticas para hacer más fácil dicha extracción, tal y como se muestra en la tabla 9.1.

Figura sintáctica	Figura de ALBA	Condiciones a cumplir
Sustantivos	Entidades / Clases	
'Es un'	Relación de herencia	
'Está formado por'	Relación de agregación	
Adjetivos	Campos / Elementos miembro	
Verbos	Escenario	<i>Si el sujeto de la acción es el módulo</i>
Verbos	Función miembro	<i>Si el sujeto de la acción es una pantalla. Si dicha acción es suficientemente compleja como para necesitar la intervención del usuario, se puede crear una nueva pantalla.</i>
Adverbios condicionales	Restricción	

Tabla 9.1. Equivalencia de elementos en la captura de necesidades

## 9.2. Subsistema de desarrollo

A continuación se expone cómo se ha desarrollado ALBA al aplicarse al patrón de arquitectura propuesto PARGEN.

### 9.2.1. Representación del conocimiento

Este módulo constituye el núcleo central de la aplicación, puesto que en él se define todo el conjunto de entidades conceptuales y funcionales que posteriormente se va a manejar. De una buena definición de datos en este módulo, se derivará un desarrollo consistente en el resto de los módulos.

Este módulo se desarrolla según las tres capas de la aplicación, ya que la responsabilidad y colaboración están enfocadas tanto a extraer lo relativo al almacenamiento como a la gestión de dicho almacenamiento y a la interfaz gráfica de usuario. De forma que en este módulo se extraen objetos mensajeros, gestores y visuales. En la figura 9.4 se pone de manifiesto el desarrollo en multicapa con los objetos particulares de cada capa.

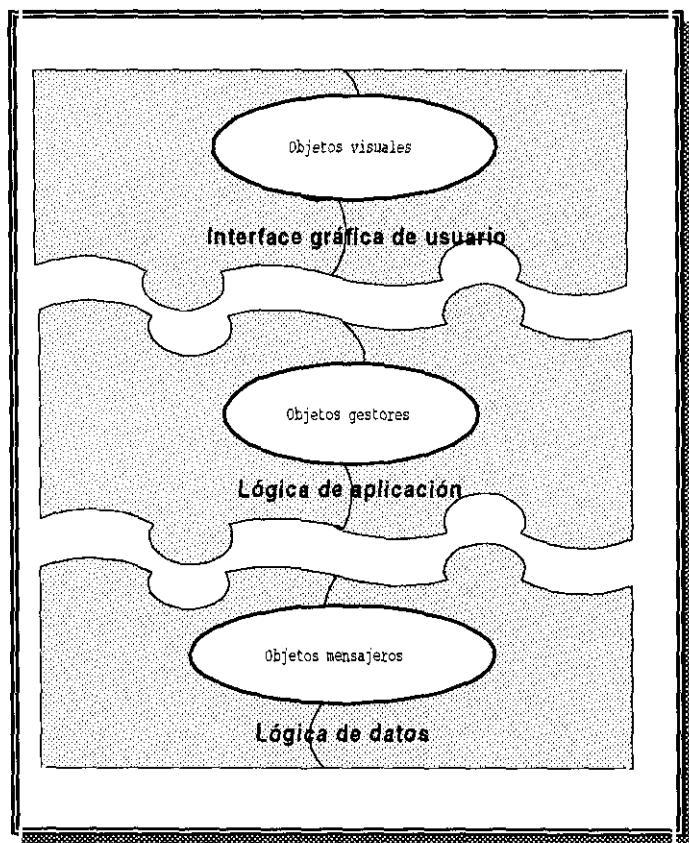


Figura 9.4. Modelo de tres capas de la aplicación

La responsabilidad y colaboración en este módulo se incluyen completamente, es decir, cubren todas las fases de cada patrón de diseño de ALBA. En la tabla 9.2, se pueden observar los patrones que se han aplicado para este módulo. Este va a ser el único que va a tener completas todas las fases de cada uno de los patrones de diseño que integran el lenguaje. Como veremos más adelante, el resto de módulos aplican los patrones tan sólo parcialmente, es decir, sin contemplar todas sus fases.

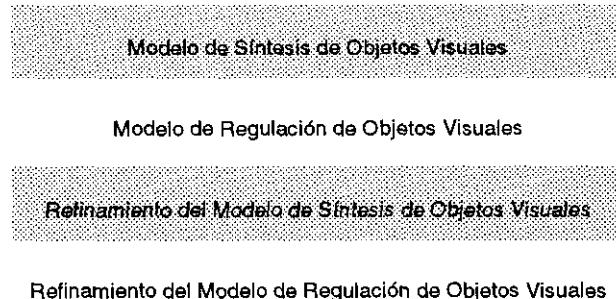


Tabla 9.2. Responsabilidad y colaboración para el módulo de representación del conocimiento

En el anexo que se presenta, se van detallando cada uno de los modelos con sus refinamientos, y dentro de éstos las fases y subfases de que consta.

## 9.2.2. Validación y verificación de la base de conocimiento

La responsabilidad y colaboración del módulo de validación y verificación están encaminadas a elaborar las funciones que aseguren que los datos que introduzca el usuario sean correctos.

El módulo de validación y verificación se plantea desde el punto de vista de la gestión de la información, que ya ha sido almacenada y visualizada en el módulo anterior y lo que se precisa es validar los datos mediante una serie de funciones. Esta es la razón por la cual la responsabilidad y colaboración se plantean desde un punto de vista funcional.. En la tabla 9.3. se resumen el conjunto de fases y subfases por las que pasan la responsabilidad y colaboración de este módulo, y que se extenderá ampliamente en anexo.

En este caso las fases de la responsabilidad y colaboración están relacionadas con la segunda capa o capa de lógica de la aplicación, de forma que la responsabilidad se cubre con la identificación de los escenarios y operaciones y la colaboración se cubre con la definición de los diagramas de escenarios y los diagramas de flujo de datos. Este módulo pone de manifiesto que el patrón propuesto responde bien para el caso de que se necesite desarrollar meramente aspectos funcionales.

Modelo de Síntesis de Objetos Visuales	Fase Funcional	Identificación de escenarios
	Fase de Activación	Construcción del diccionario de escenarios
		Identificación de las clases gestoras
		Construcción del diccionario de clases
		Construcción del diagrama de asociaciones
Modelo de Regulación de Objetos Visuales		Construcción de diagramas de asociaciones funcionales
		Identificación de operaciones
Refinamiento del Modelo de Regulación de Objetos Visuales		Construcción de los diagramas de Escenarios
		Construcción de los diagramas de Flujo de datos

Tabla 9.3. Responsabilidad y colaboración para el MVVBC

El MVVBC está formado por un escenario único, que a su vez se divide en diferentes operaciones dependiendo de la base de conocimiento de que se trate. ALBA, en este caso, está centrado en las funciones propias de validación y verificación, necesitando tan sólo que se desencadene su acción mediante el pulsado de un botón por parte del usuario. Por otra parte, todas las operaciones del módulo de validación y verificación emplean las clases de almacenamiento ya creadas en el modelo de síntesis de objetos visuales. De esta forma, el lenguaje queda bastante reducido al no tener que crear ni clases mensajeras ni objetos visuales.

Puesto que la validación y verificación van a ser diferentes según se lleve a cabo sobre un tipo u otro de unidades del conocimiento, se ha convenido desarrollar, en el anexo, el modelo de síntesis de objetos visuales al principio, al ser común tanto para reglas de producción, como para marcos de clasificación o marcos causales. Y tanto el modelo de regulación de objetos visuales como el refinamiento del modelo de regulación de objetos visuales, se ha desarrollado diferente para cada unidad de conocimiento. El módulo de validación y verificación del conocimiento emplea en los tres casos, los cuatro analizadores, pero cada analizador presenta un comportamiento distinto según a qué unidad del conocimiento se aplique.

## 9.2.3. Representación de la incertidumbre

En este módulo se han aplicado tres métodos de manejo de la incertidumbre, lo que supone que tanto la responsabilidad como la colaboración se han elaborado de diferentes formas. Por un lado la aplicación de los métodos de probabilidad subjetiva y redes bayesianas aplican ALBA de una forma paralela y por otro lado el método de los factores de certeza lo aplican de una forma totalmente distinta.

### 9.2.3.1. Probabilidad subjetiva y redes bayesianas

Estos dos métodos de manejo de la incertidumbre presentan un comportamiento parecido, puesto que ambos desarrollan completamente todas las fases de ALBA; de lo que deducimos que existe una contribución a las tres capas de la aplicación. Tan sólo encontramos una particularidad en la colaboración, y es que sólo se desciende un nivel en el refinamiento del modelo de síntesis de objetos visuales, puesto que en este caso tan sólo se trata de elaborar una pantalla que permita la introducción de probabilidades; esta pantalla depende directamente de la aplicación, y no tiene subopciones que permitan la navegación a través de ella, con lo cual tan sólo se crean los diagramas de interfaz en fase I. Este caso particular dispone tan sólo de un escenario, que se dispara cuando el usuario pide abrir el editor de probabilidad, desencadenándose la funcionalidad contenida en cada uno de los analizadores de que consta cada caso, diferentes para la probabilidad subjetiva y redes bayesianas. En la tabla 9.4 se muestra la única subfase de que dispone el refinamiento del modelo de síntesis de objetos visuales.

### 9.2.3.2. Factores de certeza

El método de manejo de factores de certeza, presenta ciertas diferencias a los casos anteriores puesto que no necesita una pantalla de entrada de datos propia, sino que emplea la clase visual proporcionada por el editor de reglas, añadiéndole un control de edición donde el usuario puede introducir el valor del factor de certeza. De esta forma, el factor de certeza se considera como elemento miembro de una clase ya creada en el módulo de representación del conocimiento para reglas de producción.

En este caso se contribuye a las tres capas de la aplicación, aunque la capa de la interfaz gráfica de usuario es poco significativa comparado con el caso anterior. En el modelo de síntesis de objetos visuales no existe ni fase de transporte ni fase funcional, porque las clases mensajeras son las extraídas en el módulo antes mencionado, y no existe una interacción funcional directa con el usuario, es decir, no existen escenarios, sino que cuando el usuario introduce los valores de los factores de certeza, se desencadenan internamente operaciones de grabado, pero que son totalmente transparentes al usuario.

En la tabla 9.5. aparece una muestra de las fases que constituyen la responsabilidad y colaboración de este caso particular del módulo de representación de la incertidumbre.

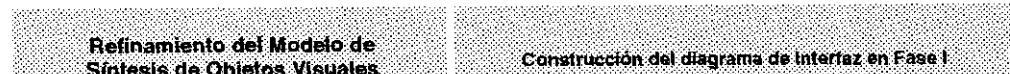


Tabla 9.4. Fases de PARSOV para el caso de probabilidad subjetiva y redes bayesianas

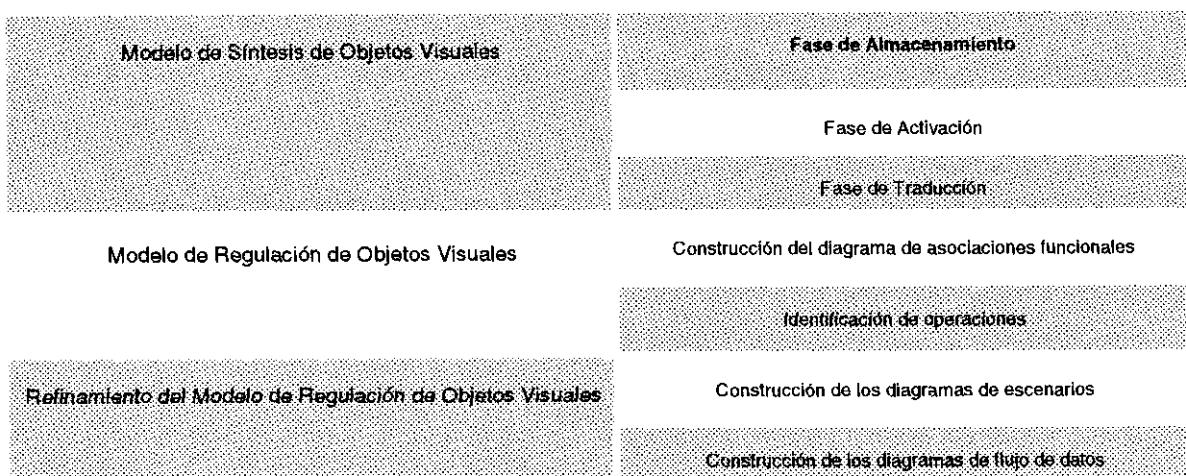


Tabla 9.5. Responsabilidad y Colaboración para el caso de factores de certeza

## 9.3. Subsistema de ejecución

### 9.3.1. Presentación del conocimiento

Este módulo presenta los cuatro patrones de diseño totalmente desarrollados, como se muestra en la tabla 9.2, al igual que pasaba con el módulo de representación del conocimiento. Aunque el volumen de la responsabilidad y colaboración de este módulo no es tan amplio como en el módulo antes citado, es necesario actuar sobre las tres capas de la aplicación, puesto que existen nuevas entidades de almacenamiento no existentes hasta ahora, nuevas clases gestoras que las manejan y una interfaz gráfica de usuario para mostrar el resultado. Este módulo también presenta una funcionalidad que es necesario desarrollar en el modelo de regulación de objetos visuales y su refinamiento. El refinamiento del modelo de síntesis de objetos visuales presenta tan sólo el diagrama de interfaz en fase I, puesto que no existe navegación de clases visuales, este módulo es meramente de presentación de información en un solo nivel.

### **9.3.2. Propagación del conocimiento**

En este módulo, la responsabilidad presenta todas las fases de que consta ALBA para la responsabilidad, sin embargo, la colaboración presenta sólo uno de los refinamientos. Se obvia el refinamiento del modelo de síntesis de objetos visuales, puesto que en este caso no existe navegación por pantallas, sino que se trata de un proceso automático, que el sistema va desencadenando según se van contestando preguntas por parte del usuario.

Este módulo es fundamentalmente funcional, con lo cual, se hace hincapié en todo lo relativo a las operaciones, que son gran número de ellas y que se desencadenan por una inicial acción del usuario. En este módulo también encontramos escenarios; aunque no son demasiados, constituyen tan sólo el desencadenante inicial para que el usuario pueda poner en marcha el motor de inferencia.

### **9.3.3. Propagación de la incertidumbre**

La responsabilidad y colaboración de este módulo sigue dos patrones diferentes. Al igual que hicimos con el módulo de representación de la incertidumbre, dividimos este módulo en sus dos variantes, la de la probabilidad subjetiva y redes bayesianas, por seguir un desarrollo paralelo en cuanto al responsabilidad y la colaboración se refiere y la de los factores de certeza.

#### **9.3.3.1. Probabilidad subjetiva y Redes bayesianas**

En este caso, la responsabilidad cumple casi todas las fases de los dos patrones de ALBA, salvo la identificación de escenario, es decir, no existe un escenario concreto que dispare el proceso de propagación, sino que es a lo largo del mismo proceso de ejecución de un S.B.C. y habiendo elegido en la sesión de ejecución que se desea propagar probabilidad subjetiva o redes bayesianas, cuando se pone en marcha la propagación de probabilidades; por lo tanto en esta etapa no existe un desencadenante directo sino indirecto. El escenario que se desencadenó en el apartado anterior al hablar de la propagación de conocimientos cuando se trata de reglas, se puede aplicar aquí mismo igualmente.

En cuanto a la colaboración, se desarrolla tan sólo al diagrama de interfaz en fase I, en el refinamiento del modelo de síntesis de objetos visuales, porque no existe navegación de pantallas en este caso. Este módulo está constituido por numerosas funciones que se desencadenan paralelamente al proceso de propagación del conocimiento, con el fin de ir mostrando la propagación de las probabilidades.

En la tabla 9.6. se muestran las fases de responsabilidad y colaboración de este caso particular del módulo de propagación subjetiva y redes bayesianas. Si las fases no se encuentran detalladas en la tabla, significa que se llevan a cabo de forma completa.

### 9.3.3.2. Factores de certeza

Ahora se pone de manifiesto la responsabilidad y colaboración tal y como se describe en la tabla 9.7. La colaboración dispone sólo de uno de los refinamientos. En este caso tampoco existe fase funcional el PASOV, ya que la propagación de los factores de certeza no se desencadena por una acción directa del usuario, al igual que en caso anterior.

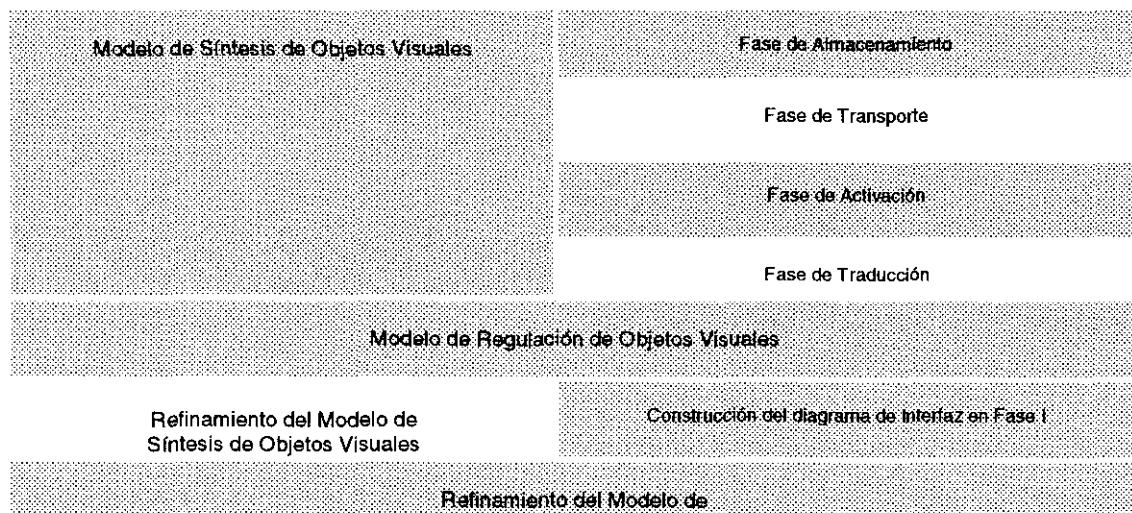


Tabla 9.6. Responsabilidad y Colaboración del módulo de propagación de probabilidad subjetiva y redes bayesianas

## 9. Integración del lenguaje de patrones con el patrón de arquitectura

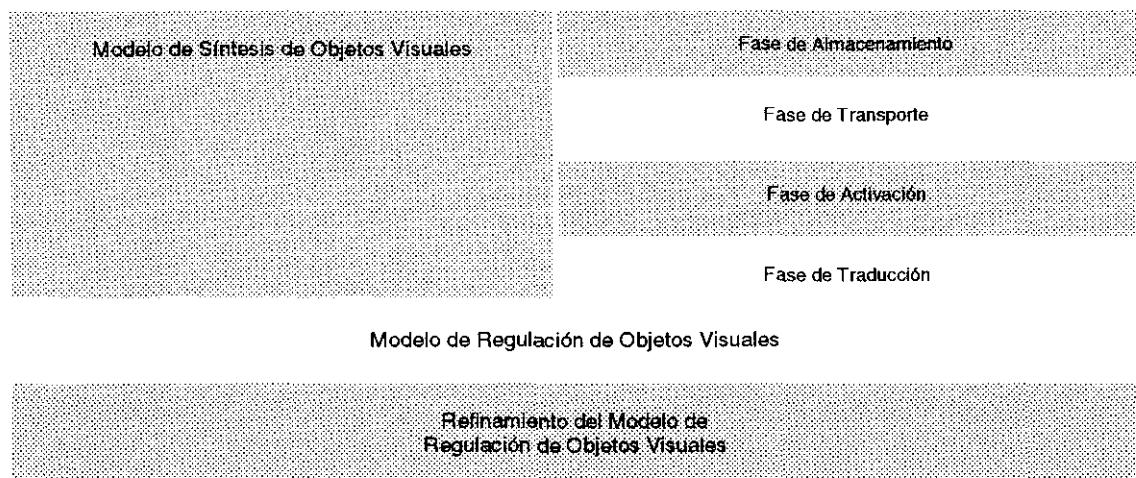


Tabla 9.7. Responsabilidad y Colaboración del módulo de propagación de factores de certeza



# **Capítulo 10**

---

## **Implementación**

### **10.1. Descripción informática**

La elaboración del caso práctico se ha realizado siguiendo el modelo en tres capas, tal y como se ha explicado en capítulos anteriores. En la figura 10.1 se muestra un diagrama de bloques que ilustra las capas de la aplicación así como el lenguaje y los protocolos de comunicación empleados.

La primera capa o capa de interfaz del usuario se ha desarrollado empleando la librería de clases *MFC 4.0* de *Microsoft*, que constituyen realmente, en el lenguaje de patrones, un marco, *framework*, al proporcionar una librería de clases genéricas para el desarrollo de aplicaciones. Hemos elegido esta librería de clases por la facilidad de manejo y las múltiples opciones de programación que proporciona.

La segunda capa viene determinada por la lógica de la aplicación, construida empleando el lenguaje C++ por ser un lenguaje estándar, orientado a objetos, actual y fácil de migrar de unas plataformas a otras.

La tercera capa constituye la lógica de los datos, almacenados en este caso concreto en la base de datos *MS-Access 7.0*, por su sencillo manejo, su versatilidad y por permitir su acceso a través de *ODBC, Open DataBase Connection*.

Los protocolos de comunicación empleados son *DLL, Dynamic Link Library*, entre la interfaz y la lógica de la aplicación, y *ODBC* entre el núcleo de la aplicación y la base de datos. Por otro lado se ha considerado interesante plasmar una cuarta capa, que realmente constituye un desdoblamiento de la segunda y se refiere a la gestión y visualización de los informes. Los informes se crean mediante la librería *Crystal Reports 6.0*, de *Microsoft*, y se comunican mediante *DLL* con el núcleo y mediante *ODBC* con el repositorio de datos.

Los requisitos informáticos mínimos recomendables de la aplicación son los siguientes.

PC: Pentium 150, 32MB RAM, Tarjeta Gráfica SVGA con Res: 1024x768.

S.O.: >=Windows 95.

Software: Conexión ODBC, MSAccess.

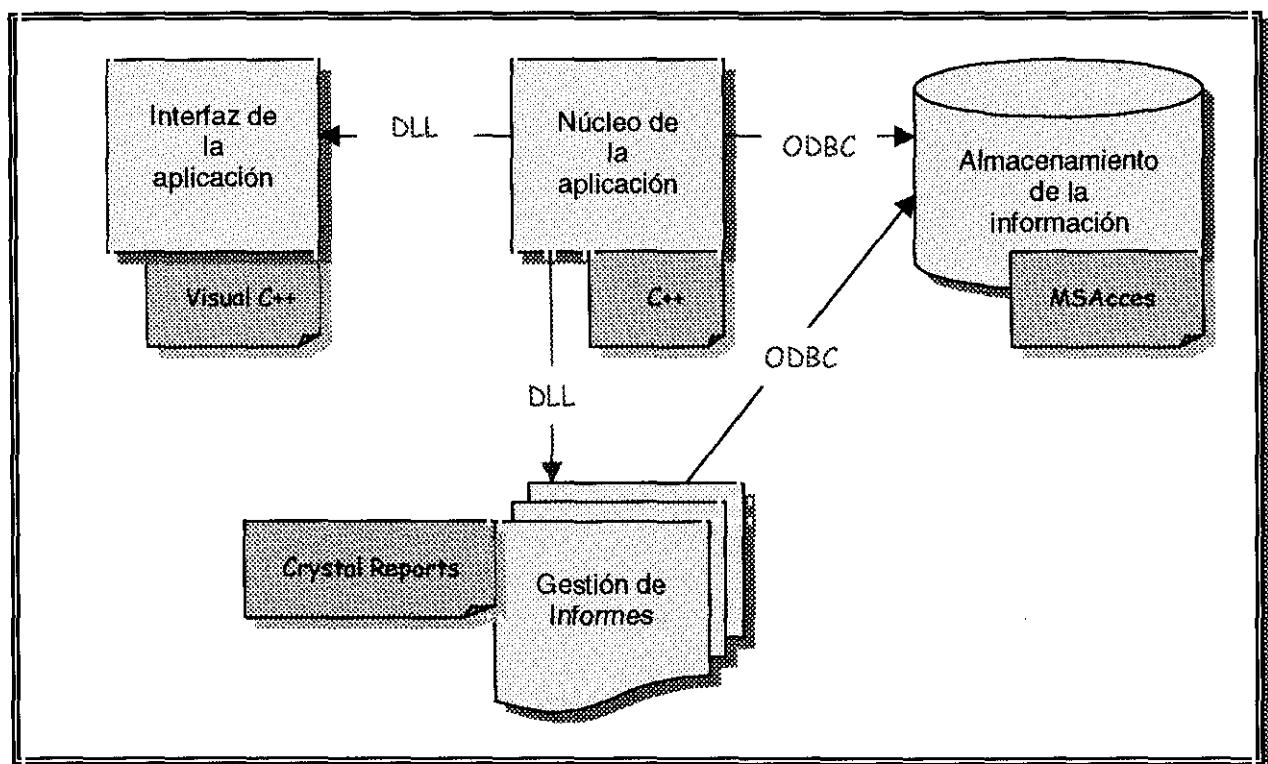


Figura 10.1. Diagrama de bloques de la aplicación

## 10.2. Desarrollo informático

En el caso práctico que hemos desarrollado, al tener dos subsistemas bien diferenciados, hemos considerado oportuno extraer una muestra lo suficientemente significativa para que el usuario se haga una idea de las pantallas y la navegación de las mismas que va a tener la aplicación.

### 10.2.1. Subsistema de desarrollo

Este subsistema se ha dividido en varias partes, con el fin de ofrecer al lector una panorámica lo más amplia posible de la aplicación desarrollada. A continuación concretamos dichas partes, para explicarlas con más detalle más adelante:

- a) Inicio de la aplicación, es decir, se trata de mostrar con qué tipo de pantallas se va a encontrar el usuario cuando quiera ejecutar la aplicación.
- b) Tipo de editores, consiste en ilustrar con qué editores se va a encontrar el usuario con arreglo a las opciones generales elegidas al configurar el S.B.C.
- c) Opciones propias de cada editor, este apartado trata de detallar la funcionalidad que proporciona cada uno de los editores presentes en la aplicación.
  - a) La configuración inicial de un S.B.C se determina en función de las opciones que aparecen cuando se crea un nuevo S.B.C. Las opciones a elegir hacen referencia a:
    - Datos generales del S.B.C. (Ver figura 10.2.)
    - Datos referentes a la representación y presentación del conocimiento, tal y como se muestra en la figura 10.3.
    - Datos referentes a la propagación de la incertidumbre, cuyas opciones a elegir, dependen de lo elegido en la pantalla anterior. (Ver figura 10.4)

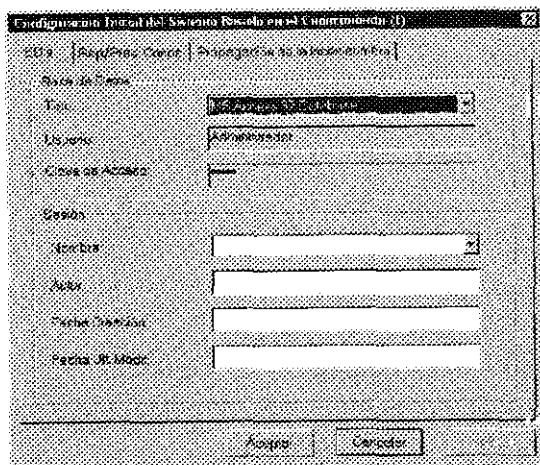


Figura 10.2. Configuración general

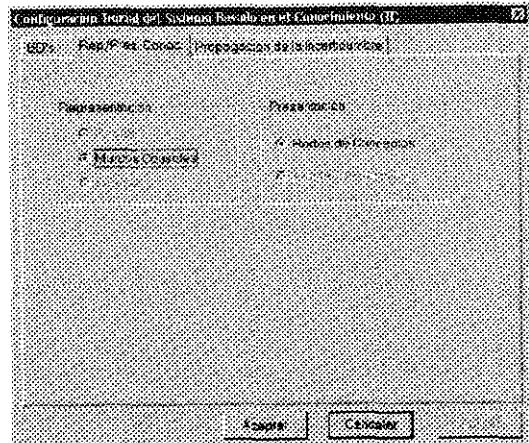


Figura 10.3. Representación /Presentación del conocimiento

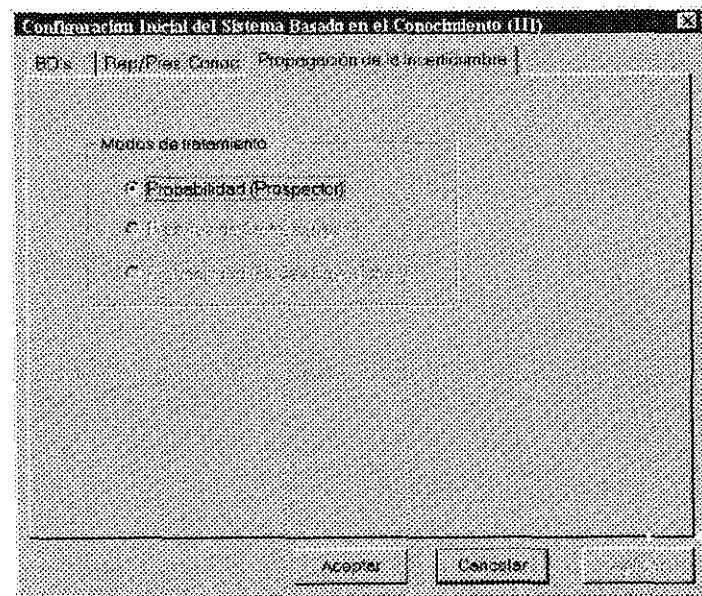


Figura 10.4. Propagación de la incertidumbre

- b) Los editores facilitados en el generador de S.B.C propuesto son numerosos, dependiendo de las opciones elegidas.
  - Si la opción seleccionada para representar el conocimiento son las reglas, automáticamente se podrá acceder al editor de reglas, de objetos y de atributos.

- Si la opción seleccionada son los marcos de clasificación, se podrá acceder al editor de marcos y al de atributos igualmente.
- Si la opción seleccionada son los marcos causales, se podrá acceder al editor de marcos causales y al de atributos.

En la figura 10.5 se muestra la opción de menú que aparece cuando se ha elegido la opción de reglas para la representación del conocimiento. De esta forma se puede ver que los editores activos son los editores de objetos y atributos, encargados del almacenamiento; el editor de reglas aparece por defecto al elegir dicha opción, de ahí que el punto de menú ‘*Editor de Reglas*’ aparezca como inactivo.

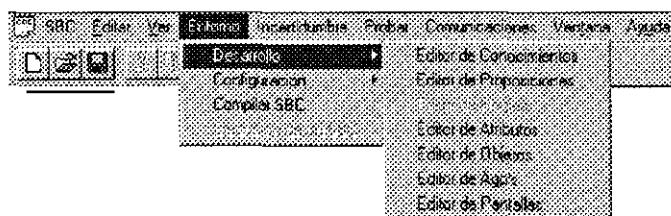


Figura 10.5. Menú ‘Entorno’ del subsistema de desarrollo

Los editores de proposiciones y conocimientos, que intervienen en la creación de la red de conceptos durante el proceso del razonamiento, aparecen como opciones de menú activas, al igual que los editores de pantallas y agrupaciones, encargados de preparar las pantallas que interaccionarán directamente con el usuario en el momento de llevar a cabo las sesiones de consultas.

En la figura 10.6. se muestra el editor de reglas, que resulta especialmente interesante por su visualizador ‘wysiwyg’, que aparece asociado a un código de colores según se van introduciendo las reglas, por otro lado presenta un visualizador de errores que muestra los errores cometidos al intentar grabar una regla.

En la figura 10.7. se muestra el editor de marcos causales, interesante por permitir la introducción del conocimiento que va a formar una red bayesiana.

Otro tipo de editores existentes, sea cual sea la opción de representación del conocimiento elegida, es el editor de agrupaciones de pantallas, asociado al editor de pantallas. A través de estos editores se generan pantallas multimedia que pueden servir de ayuda a la hora de desencadenar una explicación sobre una determinada unidad de conocimiento, o bien cuando se pone en marcha el motor de ejecución, cuando se generan las

preguntas al usuario. En la figura 10.8. se ilustra el editor de agrupaciones para hacerse una idea de su potencia.

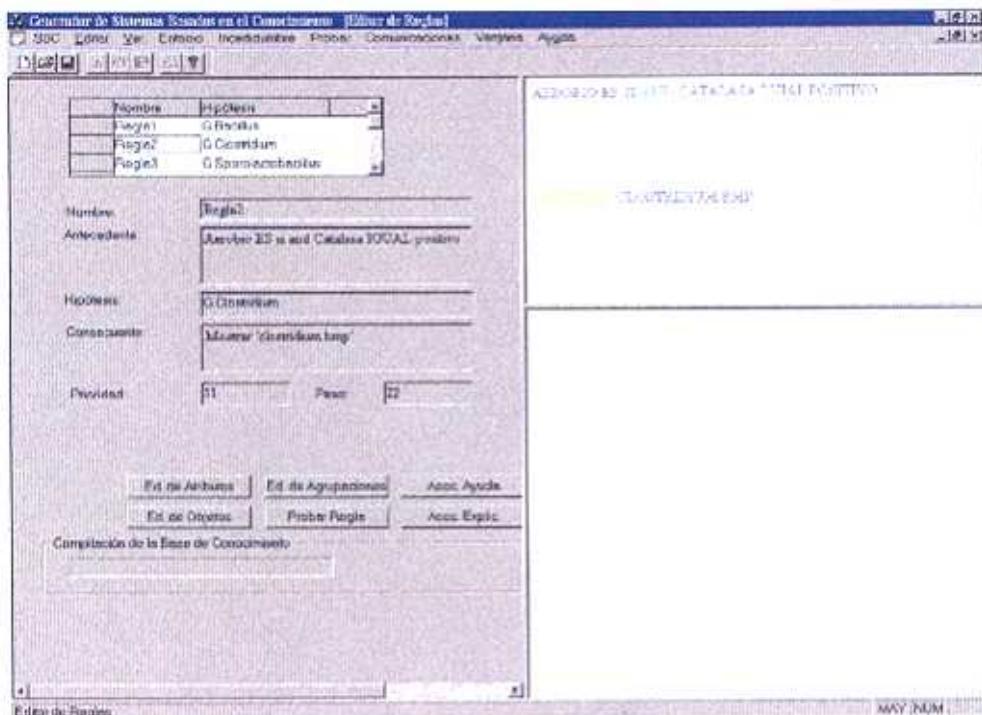


Figura 10.6. Editor de reglas

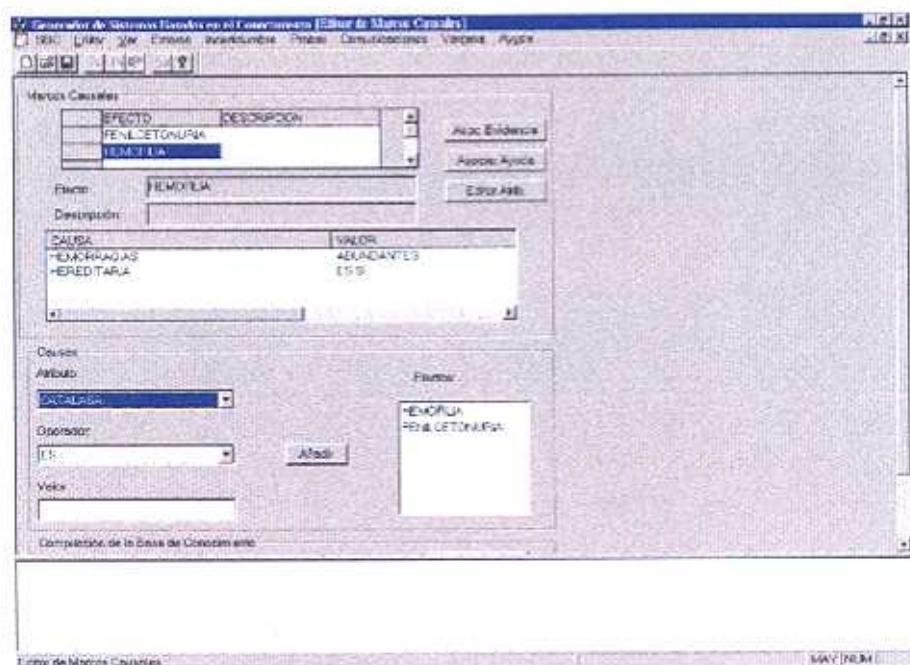


Figura 10.7. Editor de marcos causales

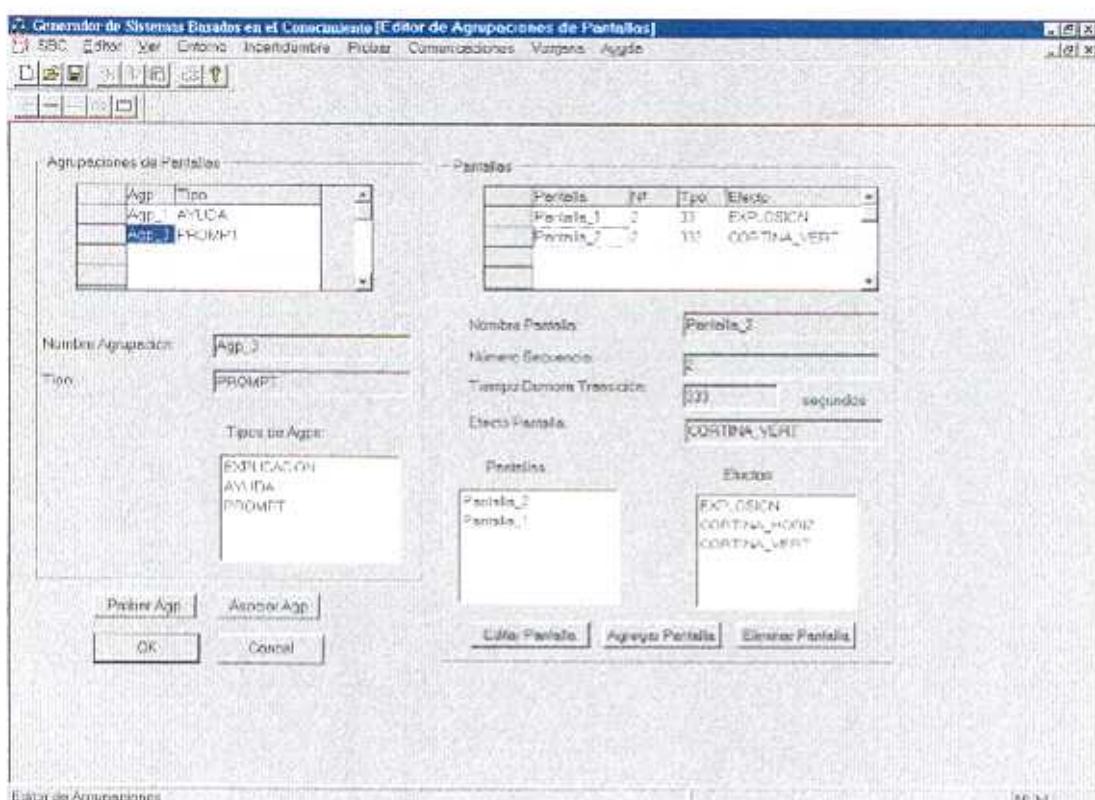


Figura 10.8. Editor de Agrupaciones.

Por otra parte, dependiendo de las opciones elegidas para la representación del conocimiento y la propagación de la incertidumbre, se activan unas u otras para la presentación del mismo, tal y como detallamos a continuación.

- Si la opción seleccionada para representar el conocimiento son las reglas, se puede presentar el conocimiento como árboles de decisión y como redes de conceptos. Si se eligen las redes de conceptos, automáticamente se activan los editores de proposiciones y de conocimiento, que se ilustran en las figuras 10.9 y 10.10.
- Si la opción seleccionada para representar el conocimiento son las reglas, se activa el editor de probabilidades y la introducción de los factores de certeza. En la figura 10.11, se ilustra el editor de probabilidades subjetivas, con su parte cuantitativa y su parte gráfica. Este editor resulta interesante, puesto que en él se muestra la regla una vez eliminados los paréntesis y los operadores lógicos: 'O', preparada para llevar a cabo la introducción de las probabilidades y las medidas de suficiencia y necesidad. Por otro lado, este editor lleva incorporado un compilador de errores, de forma que cuando se introducen probabilidades o medidas erróneas, se avisa al usuario, localizando la probabilidad errónea.

- Si la opción seleccionada son los marcos causales, se activa el editor de probabilidades para llenar la red bayesiana que se formará a la hora de propagar el conocimiento. En la figura 10.12. se ilustra el editor de probabilidades bayesianas.

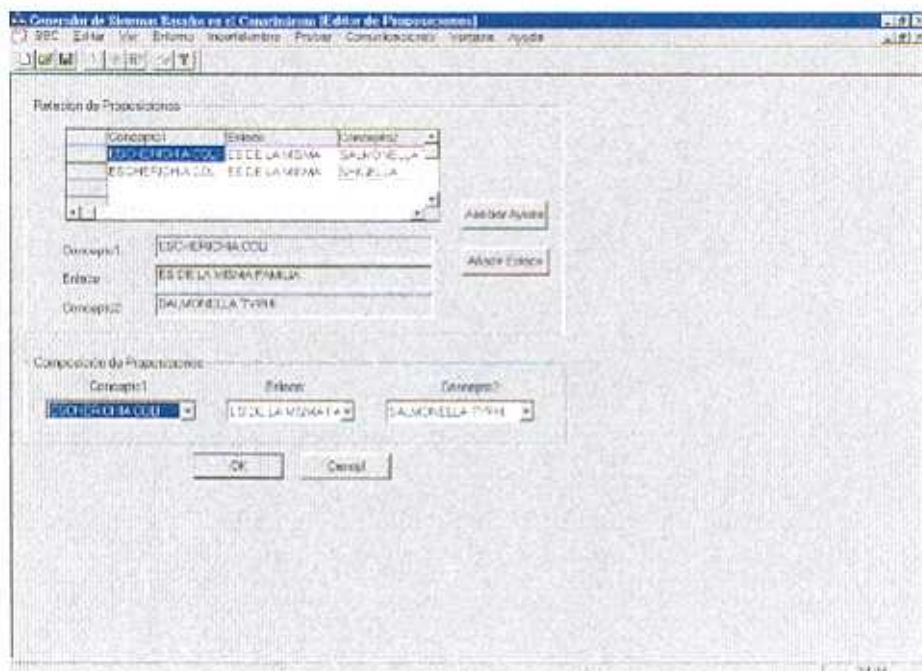


Figura 10.9. Editor de Proposiciones

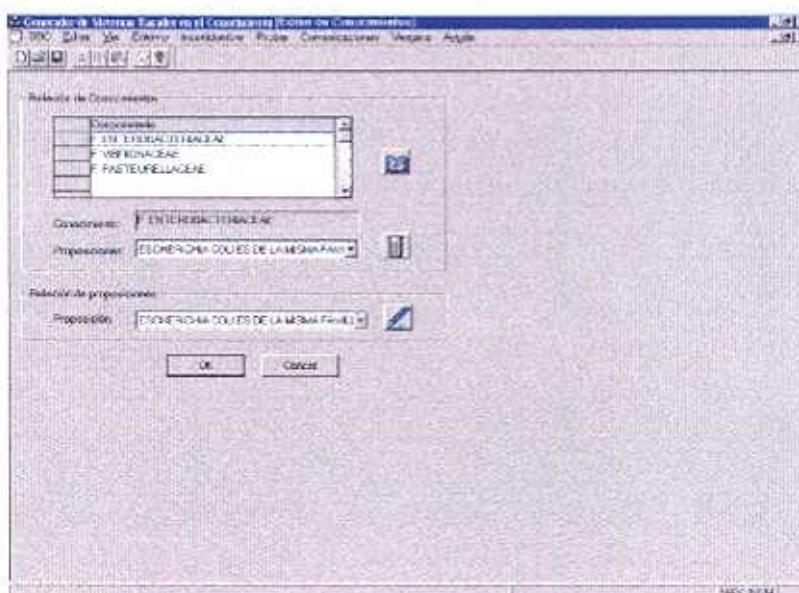


Figura 10.10. Editor de Conocimientos

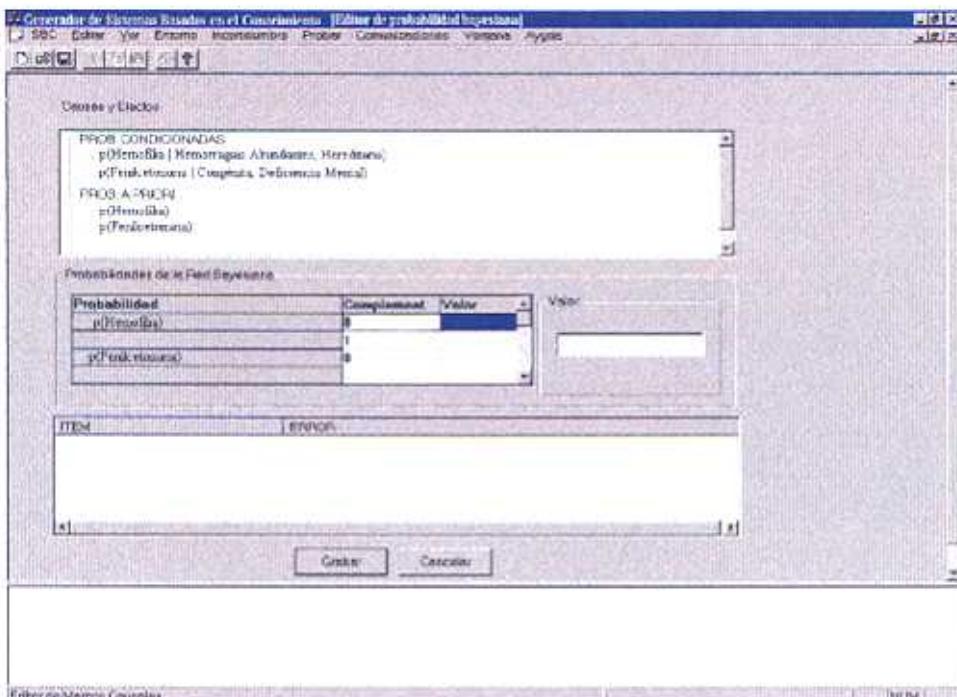


Figura 10.11. Editor de Probabilidad bayesiana

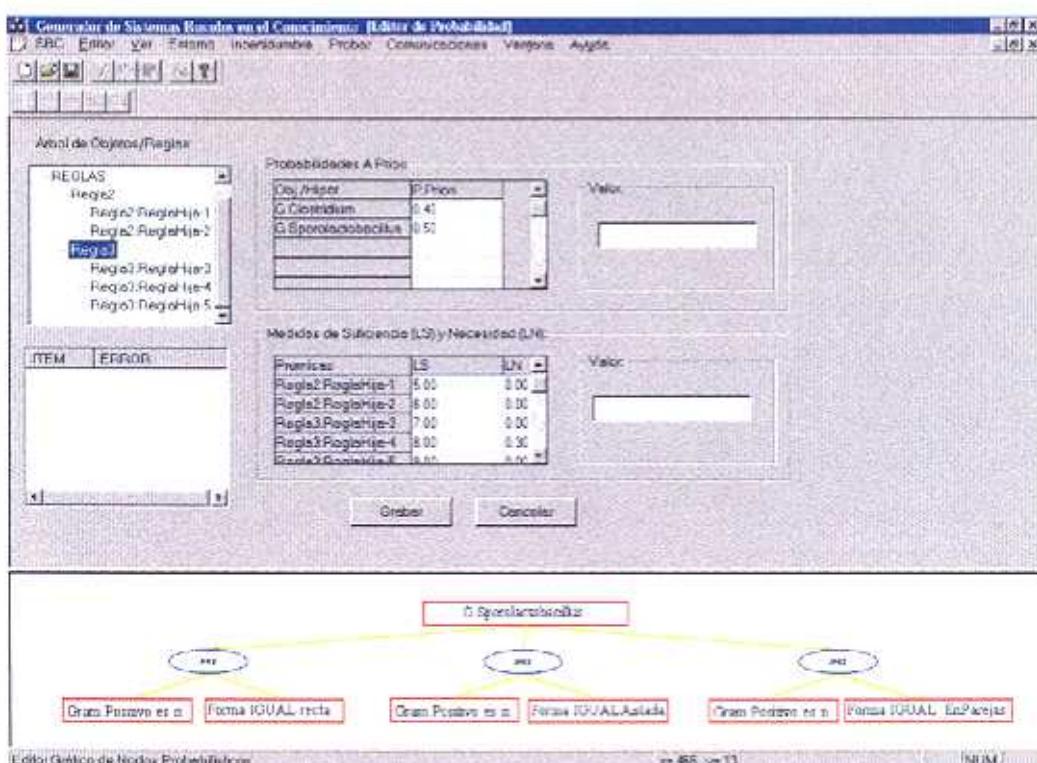


Figura 10.12. Editor de Probabilidad subjetiva

- c) Cada editor tiene a su vez, una serie de opciones que permiten la realización de consultas, la generación de informes, así como todo tipo de operaciones relacionadas con el mantenimiento de cada unidad del conocimiento. En las figuras 10.13 y 10.14 mostramos sólo algunas de las opciones existentes. Hemos elegido la opción de hacer búsquedas y la de imprimir, para el caso del editor de marcos causales.

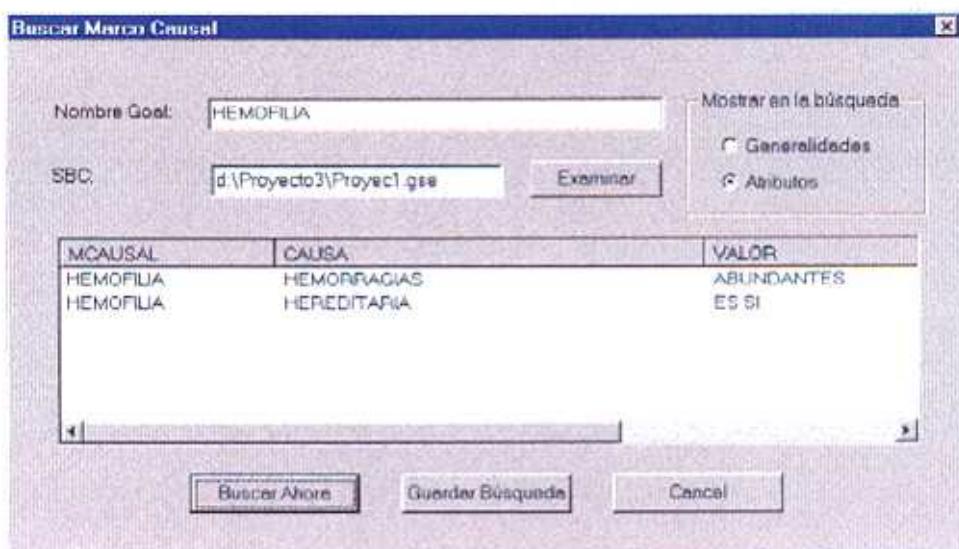


Figura 10.13. Opción 'Buscar' del Editor de marcos causales

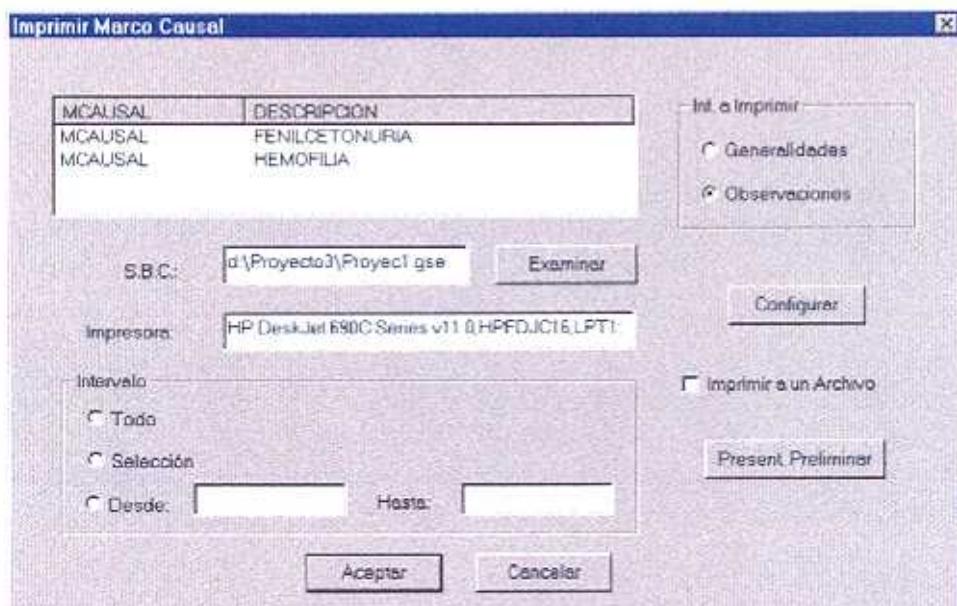


Figura 10.14. Opción 'Imprimir' del Editor de marcos causales

En el subsistema de desarrollo existe una opción disponible si se desea simular la ejecución de un S.B.C., de una pantalla o de una agrupación de pantallas. En la figura 10.15 se puede apreciar el aspecto de dicha opción.

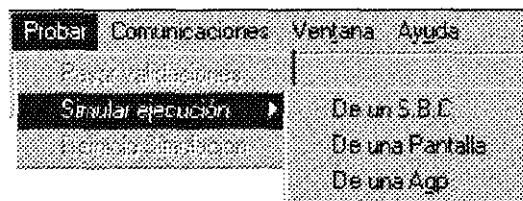


Figura 10.15. Opción ‘Probar’ del Subsistema de desarrollo

## 10.2.2. Subsistema de ejecución

El prototipo de este subsistema se ha dividido en varias partes:

- Ilustrar la configuración inicial de este subsistema, las opciones que el usuario puede elegir, y, con arreglo a ello, qué le va a aparecer en la aplicación. La configuración inicial viene determinada por tres pantallas que se muestran en las figuras 10.16, 10.17 y 10.18. En la figura 10.16 se introducen o visualizan los datos generales de la sesión de ejecución. En la figura 10.17 se especifica el tipo de presentación del conocimiento deseado y se concreta si se dispone de datos para empezar la sesión. Si no se conocen datos, el motor de ejecución se basará en prioridades, pesos o bien en otro tipo de criterios que ya fueron especificados anteriormente. Si se conocen datos, se da paso a una nueva pantalla, tal y como se detalla en la figura 10.18, donde se especificarán los datos conocidos.
- Ilustrar el proceso de ejecución, mediante el cual se detalla un fragmento del menú del subsistema de ejecución en la figura 10.19 y a continuación especificamos, en el apartado siguiente, algunos pseudocódigos concretos.

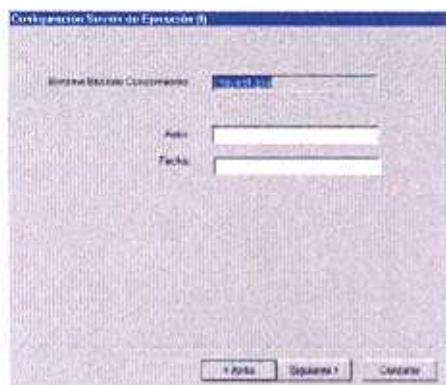


Figura 10.16. Configuración Inicial (I)

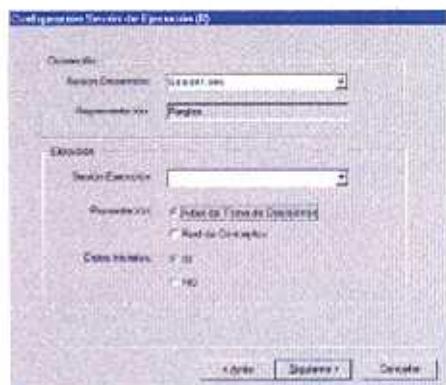


Figura 10.17. Configuración Inicial (II)

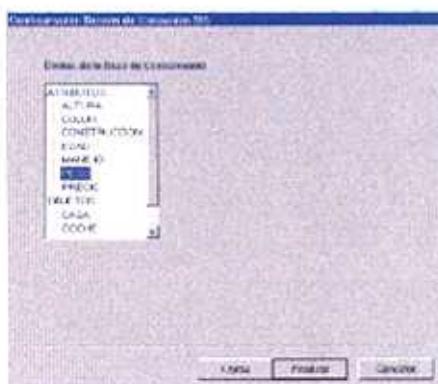


Figura 10.18. Configuración Inicial (III)

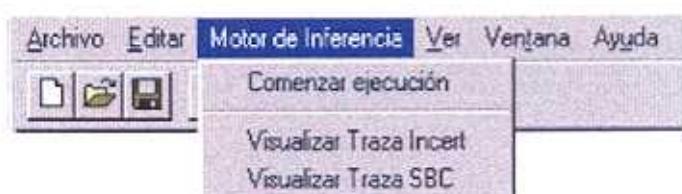


Figura 10.19. Opción de menú 'Motor de Inferencia' del subsistema de ejecución

### 10.2.3. Pseudocódigos

En este apartado se muestran algunos pseudocódigos de funciones que nos han parecido más interesantes, por su relevancia en la aplicación y por particularidades que presenta la programación de las mismas. Hemos elegido algoritmos que forman el analizador de coherencia total.

En primer lugar, en la tabla 10.1, hacemos una descripción del algoritmo que transforma los marcos causales en nodos y aristas, con el fin de llevar a cabo la construcción de la red bayesiana. Este algoritmo junto con el siguiente, son los encargados de asegurar la coherencia de la red bayesiana, con el fin de evitar la existencia de ciclos, puesto que el algoritmo de propagación elegido será válido para poliárboles.

<i>Función: Asignar nodos y aristas a los marcos causales de la BC</i>
<i>Condiciones iniciales</i>
NumMC: N° de Marcos Causales MC: Marcos Causales Parte de los Marcos Causales
Tablas usadas: MNOD_DESCR, MARI_MENSA
<b>Bucle a NumMC</b> if Existe el MC como nodo then continuar else Asignar nº de nodo (Nodo Final) al MC Grabar en MNOD_DESCR endelse endif Leer NumCausas del MC <b>Bucle a NumCausas</b> Asignar nº de nodo (Nodo Inicial) a la Causa Grabar en MNOD_DESCR if Existe la arista: Nodo Inicial → Nodo Final then continuar else Grabar arista en MARI_MENSA endif endbucle endbucle

Tabla 10.1. Pseudocódigo de la función ‘Asignar nodos’

Por otro lado, en la tabla 10.2, se muestra el algoritmo que comprueba si existen ciclos en la base de conocimiento formada por marcos causales, con el fin de que la red bayesiana sea consistente.

Función: Analizar si existen ciclos en la BC formada por los Marcos Causales*Condiciones Iniciales:*

NumNodosNSP  
 NumPadres, NumPadres1, NumHijos  
 NodosNSP  
 Padres  
 Camino[n]  
 Tablas usadas: MNOD\_DESCR, MARI\_MENSA

```

Recoger nº de nodos sin padres y guardarlo en NumNodosNSP
Asignar memoria dinámica
Recoger nodos NSP y guardarlos en NodosNSP
Bucle a NumNodosNSP (desde 0 hasta NumNodosNSP en i)
  Camino[0]=NodosNSP[i]
  Longitud=1;
  RecogeCamino(Camino[0], Longitud, Camino)
  Bucle a Longitud (desde Longitud hasta 0 en k)
    Recoge nº de padres (de Camino[k]) en NumPadres
    if NumPadres < 2
      then Continuar
    else Recoge padres (de Camino[k]) en Padres
      Bucle a NumPadres (desde 0 hasta NumPadres en j)
        Recoge nº de padres (de Padres[j]) en NumPadres1
        Mientras NumPadres1 > 0
          RecogePadres (de Padres[j]) en Padres1
          Bucle a NumPadres1 (desde 0 hasta NumPadres1 en p)
            Recoge nº de hijos (de Padres1[p]) en NumHijos
            if NumHijos > 1
              then Recoge hijos (de Padres1[p]) en Hijos
                Bucle a NumHijos (desde 0 hasta NumHijos en q)
                  if (Camino[k-1]=Hijos[q])
                    then Continuar
                  else if (Camino[k]=Hijos[q])
                    then Existe un bucle
                      endBucle
                    endif
                  endif
                endBucle
              endMientras
            endBucle
          endif
        endBucle
      endBucle
    endBucle
  endBucle

```

Tabla 10.2. Pseudocódigo de la función 'Comprobar existencia de ciclos'

Por último, en la tabla 10.3 se ilustra la elaboración del algoritmo que recoge un camino posible a partir de un nodo inicial. El interés de este algoritmo radica en el carácter recursivo del mismo, ya que se debe ir acumulando los diferentes nodos con el fin de construir un camino concreto.

Función: Recoger un posible camino a partir de un nodo inicial

*Condiciones Iniciales:*

Longitud=1

Camino → Trayectoria a partir de un nodo inicial hasta un nodo final

Camino[0] → Primer nodo de la red de conceptos

NumHijos

Hijos → Conjunto de nodos hijos de un nodo previamente especificado

Recoge nº de hijos (de Camino[0]) en NumHijos

If NumHijos < 2

then salir

else recoge hijos (de Camino[0]) en Hijos

Camino[longitud] = Hijos[0]

NumHijos=0

Recoge nº de hijos (de Camino[Longitud] en NumHijos

Mientras NumHijos > 0

Longitud=Longitud+1

Recoge Hijos de Camino[Longitud] en Hijos

Camino[Longitud] = Hijos[0]

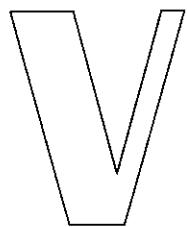
Longitud=Longitud+1

Recoge nº de hijos de Camino[Longitud] en NumHijos

EndMientras

endIf

Tabla 10.3. Pseudocódigo de la función ‘Recoge Camino’



# CONCLUSIONES

# **Capítulo 11**

---

## **Conclusiones**

### **11.1. Aportaciones de la tesis**

En esta tesis hemos propuesto por un lado un *Patrón de Arquitectura para la construcción de Generadores de Sistemas Basados en el Conocimiento (PARGEN)* con aplicación en áreas biológicas. Por otro lado, con el objetivo de formalizar las labores de análisis y diseño necesarias en la programación de toda herramienta informática, se ha desarrollado un *Lenguaje de Patrones para la Síntesis y Regulación de Objetos Visuales (ALBA)* que facilita dichas tareas, gracias a la integración en él de cuatro patrones de diseño genéricos orientados a objeto. La aplicación de dichos patrones de diseño al generador de S.B.C. ha proporcionado la creación de otros patrones de diseño específicos a modo de implementación práctica de los anteriores.

El lenguaje de patrones propuesto presenta como novedad intrínseca el basarse en los mecanismos que emplea la naturaleza para perpetuarse, habiéndose establecido un fiel paralelismo

entre las unidades que intervienen en la biosíntesis de proteínas y su funcionalidad y los elementos que intervienen en cada uno de los patrones de diseño componentes de dicho lenguaje, amén de las tareas a cubrir.

El patrón de arquitectura aportado proporciona una separación entre las diferentes tareas a realizar, en un generador de S.B.C. aplicado a biología, diferenciándose una serie de módulos para cada una de ellas. Así mismo, se agrupan en analizadores aquellas subtareas existentes dentro de cada módulo y que presentan un objetivo común, haciendo mucho más fácil la modularidad de la herramienta y su reutilización.

La perfecta integración de PARGEN y ALBA se ponen de manifiesto en los resultados de esta tesis, plasmados en el anexo que se acompaña. Dichos resultados constituyen un caso práctico que muestra todo el ciclo de vida de software para cada uno de los módulos componentes de la arquitectura aportada por separado, habiéndose notado diferencias a la hora de aplicar el lenguaje de patrones a cada uno de ellos.

El caso práctico, construcción de un generador de un S.B.C., presentado en el anexo, reafirma que el lenguaje de patrones hace más hincapié en unos patrones de diseño que en otros, dependiendo de las características que tenga el problema a resolver, observándose que es igualmente válido tanto en áreas en las que predomine la componente funcional, como en las que predomine la componente conceptual o visual.

El análisis de los resultados obtenidos arroja una serie de conclusiones. En primer lugar extraemos que el lenguaje de patrones ALBA denota las propiedades que detallamos a continuación que la definen y estructuran.

- **Encapsulamiento y especialización.** ALBA presenta una serie de principios esenciales a partir de la biosíntesis de proteínas: el empaquetamiento críptico de la información, materializado en el encapsulamiento y la división de funciones, puesta de manifiesto en la construcción de un modelo de tres capas, que ha facilitado la creación de objetos especializados en determinadas tareas. Dicho modelo separa en las siguientes tres capas el desarrollo informático: almacenamiento, control y visualización, mediante la especialización de objetos: objetos mensajeros, gestores y visuales.
- **Intuitiva.** ALBA proporciona una secuencia de fases y una notación particular que resultan ambas totalmente intuitivas en contraste con la terminología que con frecuencia ofrecen metodologías de análisis y diseño informático.
- **Responsabilidad-Colaboración.** Cada uno de los patrones de diseño que componen ALBA acentúa un aspecto diferente en el desarrollo de un sistema informático. Se concluye pues que este lenguaje cubre perfectamente la interacción responsabilidad-colaboración, a través

de los cuatro patrones de diseño que lo integran. Los dos primeros (PASOV y PAROV) completan la responsabilidad conceptual y funcional respectivamente. Los dos siguientes (PARSOV Y PARROV) culminan la colaboración conceptual y funcional respectivamente. De esta forma en los resultados se puede ver claramente cómo, dependiendo del módulo de que se trate, se desarrollan más unos patrones que otros, según qué aspectos acentúe dicho módulo.

- **Independiente de secuencia.** Los cuatro patrones de diseño que componen el lenguaje de patrones, pueden aplicarse de forma **secuencial**, uno a continuación del anterior, o bien de forma **autónoma**. Se ha observado que, incluso se puede aplicar uno de los patrones de diseño de manera incompleta, haciendo uso tan sólo de aquellas fases que subrayen un determinado aspecto del módulo que se esté desarrollando en ese momento.
- **Orientado a objetos.** El lenguaje de patrones, y como consecuencia los patrones de diseño que lo integran, es acorde con el enfoque orientado a objetos, ya que cumple los cuatro principios en que éste se basa:
  - **Abstracción:** ALBA es capaz de detectar la funcionalidad principal que precisa todo desarrollo informático y lo plasma en el desarrollo de tres tipos de clases abstractas: clases mensajeras, gestoras y visuales.
  - **Encapsulamiento:** ALBA oculta la información extraída de los repositorios de datos, proporcionando diferente nivel de acceso a la misma.
  - **Herencia:** ALBA favorece mecanismos por los cuales se pueden crear clases derivadas a partir de las clases principales del lenguaje de patrones, con el fin de compartir sus datos y funciones miembro.
  - **Polimorfismo:** ALBA dispone de funciones miembro referenciadas de la misma forma, que dependiendo desde qué objeto se estén invocando, cumplen una misión u otra.
- **Almacenamiento relacional / Objetos.** El lenguaje de patrones, ALBA, proporciona pautas concretas para realizar el mapeo de objetos a un almacenamiento basado en tablas.
- **Construcción de la IGU.** ALBA aporta los recursos necesarios para conectar las entidades extraídas en las primeras fases del análisis con la construcción de la I.G.U.

En cuanto al patrón de arquitectura PARGEN, podemos deducir como principal característica la siguiente:

- **Propósito general.** Se concluye que el patrón de arquitectura, PARGEN, proporciona la división en subsistemas y módulos para un generador de S.B.C. independientemente del área

en que se aplique. Los subsistemas y módulos que lo componen, así como los analizadores de éstos, pueden aplicarse a otras áreas, no sólo las áreas biológicas. Luego el patrón de arquitectura propuesto es de propósito general.

Como consecuencia de la aplicación a un caso práctico concreto del lenguaje de patrones y el patrón de arquitectura, extraemos las siguientes conclusiones:

- **Integración de patrones.** PARGEN y ALBA, forman un todo que, si lo engarzamos con la librería de clases MFC 4.0 de Microsoft, empleada para la construcción de la interfaz de usuario, observamos que realmente lo que se presenta es la integración de tres elementos: el lenguaje de patrones, con sus patrones de diseño componentes, el patrón de arquitectura y el marco representado en las MFC.
- **Desarrollo informático consistente.** La integración de los tres patrones ha desembocado en el desarrollo de una herramienta informática plenamente funcional y consistente.
- **Generador de S.B.C.** Se concluye que la arquitectura para el generador de S.B.C. es una propuesta de herramienta para la toma de decisiones, que engloba en sí misma mucha funcionalidad. Permite la introducción del conocimiento de tres formas, favorece la propagación de la incertidumbre siguiendo tres esquemas diferentes y proporciona dos mecanismos de propagación del mismo dependiendo del nivel de información del experto. Si tiene un completo conocimiento del área a tratar son más adecuados los árboles de decisión si, por el contrario, dispone de ciertas lagunas en su conocimiento, se recomiendan las redes de conceptos.

## 11.2. Líneas futuras de desarrollo

Con esta tesis se abren una serie de líneas de desarrollo a partir de lo ya construido hasta ahora. Estas líneas se refieren tanto al lenguaje de patrones como al mismo patrón de arquitectura. A continuación citamos las que merecen más atención.

- Creación de una **herramienta informática** que automatice la generación de patrones de diseño genéricos a partir de una determinada información acerca de un problema real. Esta herramienta tendría incorporado el lenguaje de patrones propuesto y vendría a ser una herramienta generadora de patrones de diseño, acordes con el problema a resolver.
- Creación de una **herramienta de diseño** que, por un lado, implemente un módulo con la captura de requisitos para generar las distintas fichas de manera automática y por otro disponga de un segundo módulo que, a su vez, implemente los cuatro patrones de diseño genéricos que integran ALBA. De esta forma, a partir de la introducción de unos requisitos

específicos, se crearán primeramente unas fichas que el segundo módulo transformará en el modelo de tres capas del lenguaje propuesto proporcionando un esqueleto programable en cualquier lenguaje de programación visual.

- Desarrollo de una **base de conocimiento híbrida** que permita la convivencia de reglas de producción, marcos de clasificación y marcos causales. Esto conlleva el hecho de dilucidar una serie de equivalencias entre las diferentes unidades de conocimiento, con el fin de crear una base de información homogénea, así como un módulo de validación y verificación para dicha base.
- Elaboración de un diccionario de **equivalencias de clases visuales**, de forma que la clasificación de estas clases realizada en PASOV y totalmente dirigida a crear la interfaz de usuario con las MFC, pueda ser trasladable a cualquier otro tipo de entorno visual.
- Introducción de un pequeño módulo de propagación de **lógica difusa**, para aquellos atributos que no se conozcan cuantitativamente sino cualitativamente.

## BIBLIOGRAFÍA

---

- [1] Acker, L.; Porter, B. "Extracting viewpoints from knowledge bases", AAAI/MIT Press Cambridge, MA. 1994.
- [2] Adams. "Fault-tolerant telecommunications patterns". *Patterns languages of program design 2*. Reading, MA: Addison-Wesley, 1996.
- [3] Akoumianakis, D.; Stephanidis, C. "Knowledge-Based Support for User Adapted Interaction Design". *Expert systems with applications*, 12(2):225-245, 1997.
- [4] Alexander, C. "A pattern language". Oxford University Press, New York, 1977.
- [5] Alexander, C. "The Timeless Way of Building". Oxford University Press, New York, 1979.
- [6] Alonso, J.; Bailador, A.; Rodriguez F.; Levy, M.; Buencuerpo, V. "Aplicación de algunos sistemas expertos a la identificación de ascidias". *Bol. R.Soc. Esp. Hist. Nat. (Sec. Biol)*, 86:1-4, 1990.
- [7] Alonso, J.; Lahoz-Beltrá, R.; Bailador, A.; Rodriguez F.; Levy, M.; Diaz-Ruiz, R. "An expert system to classify plant viruses". *Binary*, 4:195-199, 1992.
- [8] Amescúa, A. De; García Sánchez, L.; Martínez Fdez, P.; Díaz Pérez, P. "Ingeniería del Software de Gestión. Análisis y Diseño de Aplicaciones". Ed. Paraninfo, 1995.
- [9] Andersen, S.K.; Olesen, K.G.; Jensen, F.V.; Jensen, F. "HUGIN: A shell for building Bayesian belief universes for expert systems". En: *Proceedings of the 11<sup>th</sup>. International Joint Conference on Artificial Intelligence (IJCAI-89)*. Págs:1080-1085. 1989.
- [10] Bailin, S.C. "An Object-Oriented Requirements Specification Method". *Communications of the ACM*, 32(5):608-623, 1989.
- [11] Barnett, J.A., 1981: "Computational Methods for a Mathematical Theory of Evidence". *Proc. International Joint Conference on Artificial Intelligence (IJCAI-81)*. Vancouver, 1981.
- [12] Bauer, E.; Koller, D.; Singer, Y. "Update rules for parameter estimation in Bayesian Networks". *Proc. Of the 13<sup>th</sup>. Annual Conference on Uncertainty in A.I. (UAI-97)*, Págs: 3-13, Providence, Rhode Island, August, 1/3, 1997.
- [13] Beck, K.; Cunningham, W. "A laboratory for teaching object-oriented thinking". *Proc. of Object-Oriented Programming Systems, Languages and Applications (OOPSLA '89). SIGPLAN Notices*, 24(10):1-6, 1989.

- [14] **Berczuck, S.** "Finding solutions through pattern languages". *Computer*, 27(12), December, 1994. En <http://world.std.com/~berczuk/pubs/Dec94iecc.htm>
- [15] **Bittencourt, G.; Calmet, J.; Homann, K.; Lulay, A.** "MANTRA: A Multi-Level Hybrid Knowledge Representation System". *Proceedings of the XI Brazilian Symposium on Artificial Intelligence*, Fortaleza, Págs: 493-506, 1994.
- [16] **Bobrow, D.G.; DiMichiel, L.G.; Gabriel, R.P.; Keene, S.E.; Kiczales, G.; Moon, D.A.** "Common Lisp Object System". Specification: X3J13 Document 88-002R. *SIGPLAN Notices*, 23, 1988.
- [17] **Booch, B.** "Object-Oriented Design with Applications", Benjamin Cummings, 1991.
- [18] **Booch, B.; Jacobson, I.; Rumbaugh, J.** "Unified Modelling Language". Rational Software Corporation, 1997. <http://www.rational.com>
- [19] **Boufriche-Boufaïda, Z.** "A purely object-oriented approach for rule-based paradigms". *Expert Systems With Applications*, 14:483-492, 1998.
- [20] **Buchanan, B.G.; Sutherland, G.; Feigenbaum, E.A.** "Heuristic DENDRAL: A program for generating explanatory hypothesis in organic chemistry". *Machine Intelligence 4*. B. Meltzer y D. Michie (eds), Edinburg University Press, Edinburg, 1969.
- [21] **Buchanan, B.G.; Feigenbaum, E.A.** "DENDRAL and Meta-DENDRAL: Their applications dimension" *Artificial Intelligence: An International Journal*, 11:5-24, 1978.
- [22] **Budinsky, F.J.; Finnie, M.A.; Vlissides, J.M.** "Automatic code generation from design pattern". *Object Technology*, 35(2), 1996.
- [23] **Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerland, P.; Stal, M.** "Pattern-Oriented Software Architecture, A System of Patterns", New York: John Wiley & Sons, 1996.
- [24] **Casey, R.M.** "Object Mappings in a Software Engineering Project". *Software Engineering Notes. ACM SIGSOFT*, 24(1):55-58, 1999.
- [25] **Castillo, E.; Alvarez, E.** "Sistemas Expertos. Aprendizaje e incertidumbre". Ed. Paraninfo. 334 Págs. 1989.
- [26] **Castillo, E.; Gutiérrez, J.M.; Hadi, A.S.** "Sistemas Expertos y Modelos de redes probabilísticas". Monografías de la academia de ingeniería. Edita Academia de Ingeniería, (Ed) 1996.
- [27] **Chander, P.G.; Preece, A.; Grossner, C.** "Path hunter: a tool for finding the paths in a rule based systems". DAI Technical Report DAI 0592-0012, Concordia University, Montreal Quebec, 1992.
- [28] **Chander, P.G.; Radhakrishnan, T.; Shinghal, R.** "Design Schemes for Rule-Based Systems". *International Journal of Expert Systems*, 10(1):1-36, 1997.
- [29] **Charniak, E.** "A frame painting the representations of a common sense knowledge fragment". *Journal of cognitive sciences*, 1(4): 355-394, 1977.
- [30] **Christaller, T; Di Primio, F.** "The AI-Workbench BABYLON: An open and portable environment for expert systems". Academic Press, London, 1992.
- [31] **Clark, P.; Porter, B.** "Building Concept Representations from Reusable Components". *Proceedings on AI, AAAI 97*. Págs: 369-376, 1997.
- [32] **Clocksin, W.F.; Mellish, C.S.** "Programming in Prolog". Springer-Verlag, 1981.
- [33] **Coad, P.; Yourdon, E.** "Object-Oriented Analysis". Prentice-Hall, 1991.
- [34] **Coad, P.; Yourdon, E.** "Object-Oriented Design". Prentice-Hall, 1991.
- [35] **Coad, P.** "Object-oriented patterns". *Communications of the ACM*, 35(9):152-159, September, 1992.
- [36] **Cockburn, A.A.** "In search of a methodology". *Object Magazine*, July-August, Págs: 52-57, 1994.
- [37] **Coleman, D.; Arnold, P.; Bodoff, S.; Dollin, C.; Gilchrist, H.; Hayes, F.; Jeremaes, P.** "Object-Oriented Development - The Fusion Method". EnglewoodCliffs, N.J.: Prentice-Hall, 1994.

- [38] Compton, P.; Kang, B.; Preston, P; Mullholland, M. "Knowledge Acquisition Without Analysis". *Knowledge Acquisition for Knowledge-Based Systems. 7th European Workshop, EKAW '93 Proceedings*. Springer-Verlag, Berlin, Germany. Págs: 277-299, 1993.
- [39] Coplien, J.O. "Advanced C++ Programming Styles and Idioms". Addison-Wesley, Reading, Massachusetts, 1992.
- [40] Coplien, J.O. "Software Design Patterns: Common Questions and Answers", *OBJECT EXPO Conference Proceedings*, 6-10 June. SIGS Publications, New York, NY, USA; Págs: 39-42, 1994.
- [41] Coplien, James O. and Douglas C. Schmidt, "Pattern Languages of Program Design", Addison-Wesley, 1995.
- [42] Crick, F.H.C.; Barnett, L.; Brenner, S.; Watts-Tobin, R.J. "General Nature of the genetic code for proteins". *Nature*, 192(4809):1227-1232, 1961.
- [43] Crick, F. "What Mad Pursuit. A personal view of scientific discovery". Basic books. Inc. Publishers. New York, 1988.
- [44] Cuena, J., 1985. "Lógica Informática". Alianza Editorial, Madrid, 1985.
- [45] Cunningham, W. "EPISODES: A Pattern Language of Competitive Development. Part I", *Second International Conference on Pattern Languages of Programs*. Monticello, Illinois, 6-8 September, 1995.
- [46] Cunningham, W.; Beck, K. "Using pattern languages for object-oriented programs". *OOPSLA*, 1987.
- [47] Davis, R.; Buchanan, B.G.; Shortliffe, E.H. "Production rules as a representation for a knowledge-based consultation program". *Artificial Intelligence*, 8:15-45, 1977.
- [48] Davis, R. "Meta-rules: reasoning about control". *Artificial Intelligence*, 15:179-222, 1980.
- [49] Davis, R.; Buchanan, B. "Meta-level knowledge" Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project", cap. 28. B.G. Buchanan y E.H. Shortliffe (eds), Págs: 507-530, Addison-Wesley, Reading, MA, 1984.
- [50] De Miguel, A.; Piattini, M.G. "El diseño y desarrollo orientado a objeto (I)". *Chip*. Págs: 38-43. Enero, 1993; "El diseño y desarrollo orientado a objeto (II)". *Chip*. Págs: 54-56. Febrero, 1993.
- [51] DeMarco, T. "Structured Analysis and System Specification". Englewood Cliffs, N.J.: Prentice-Hall, 1978.
- [52] Díez Vegas, F.J. "Local conditioning in Bayesian Networks". *Artificial Intelligence*, 87:1-20, 1996.
- [53] Díez Vegas, F.J. "DIAVAL, a bayesian expert system for echocardiography". *Artificial Intelligence in Medicine*, 10:59-73, 1997.
- [54] Dubois, D. & Prade, H. "Consonant Approximations of Belief Functions". *International Journal of Approximate Reasoning*, 4:419-449, 1990.
- [55] Duda, R.; Gaschnig, J.; Hart, P. "Model Design in the Prospector Consultant System for Mineral Exploration". *Expert System in the microelectronic age*, 124:153-167, 1979.
- [56] Durkin, J. "Expert Systems: A view of the Field". *IEEE Expert*, Págs: 56-63 Abril, 1996.
- [57] Eguilior, S.; Bailador, A.; Sánchez, E.; Ruiz, C. "Utilización de KOD (Knowledge oriented design) en el desarrollo de una metodología para la generación de escenarios", *Sociedad Nuclear Española, 22 reunión anual*, Págs: 486-488. Santander, 1996.
- [58] Eshelman, L. "MOLE: A knowledge acquisition tool that buries certainty factors". *International Journal Man-Machine Studies*, 29:563-577, 1988.
- [59] Fang, F.A.; So, A.C.; Kreindler, R.J. "The visual modelling technique: An introduction and overview". *JOOP*, 9(4):48-57; 73, 1996.
- [60] Fichman, R.G.; Kemerer, C.F. "Object-Oriented and Conventional Analysis and Design Methodologies". *IEEE*, Octubre, Págs: 22-39, 1992.

- [61] France, R.B.; Bruel, J.M.; Larrondo-Petrie, M.M. "An Integrated Object-Oriented and Formal Modelling Environment". *JOOP*, 10(7):25-34; 50; 1997.
- [62] Frost, R. "Bases de Datos y Sistemas Expertos. Ingeniería del Conocimiento", Ed. Díaz de Santos, Madrid, 1989.
- [63] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. "Design Patterns: Abstraction and Reuse of object-oriented design", 1993.
- [64] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. "Design Patterns: Elements of Reusable Object-Oriented Software". Reading, MA: Addison-Wesley, 1995.
- [65] Gane, C.; Sarson, T. "Structured Systems Analysis", Englewood Cliffs, N.J.: Prentice-Hall, 1979.
- [66] Genesereth, M.R.; Nilsson, N.J. "Logical Foundations of Artificial Intelligence". Los Altos, CA, Morgan Kaufmann, 1987.
- [67] George, J.; Carter, B.D. "A Strategy for Mapping from Function-Oriented Software Models to Object-Oriented Software Models". *Software Engineering Notes*, 21(2):56-63, 1996.
- [68] Gorostiza, C.; Alvarez de Lara, P.; Sánchez, E.; Bailador, A.; Alguacil, R.; González, C. "Decision Support System for Industrial Accidents on Environment: SIRENAS". *The Second World Congress on Expert Systems*, Lisbon/Estoril, 1994.
- [69] Grabot, B.; Caillaud, E. "Imprecise Knowledge in Expert Systems: A simple shell". *Expert Systems With Applications*, 10(1):99-112, 1996.
- [70] Grossner, C.; Preece, A.D.; Chander, P.G. "Exploring the Structure of Rule Based Systems". *AAAI-93*. Págs: 704-709, 1993.
- [71] Grossner, C.; Chander, P.G.; Radhakrishnan, T.; Preece, A.D. "Revealing the Structure of Rule-Based Systems". *International Journal of Expert Systems*, 9(2): 255-278, 1996.
- [72] Grossner, C.; Preece, A.D.; Chander, P.G.; Radhakrishnan, T.; Hess, M. "Mixed-level knowledge representations and variable-depth inference in natural language processing", *International Journal on AI Tools (IJAIT)*, 6(4): 481-509. 1997.
- [73] Gutiérrez Llorente, J.C. "Sistemas Expertos, Grafos y Redes Bayesianas" Tesis doctoral por el Dpto. de Matemática Aplicada y Ciencias de la Computación. Universidad de Cantabria, 1994.
- [74] Haddawy, P. "Generating Bayesian Networks from Probability Logic Knowledge Bases". *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 1994.
- [75] Haddawy, P.; Jacobson, J.; Kahn, Charles E. Jr. "BANTER: A Bayesian Network Tutoring Shell", *Artificial Intelligence in Medicine*, 10:177-200, 1997.
- [76] Hardgrave, B.C. "Adopting Object-Oriented Technology: Evolution or Revolution?". *Journal of Systems Software*, 37:19-25, 1997.
- [77] Hasselbring, W.; Kröber, A. "Combining OMT with a prototyping approach". *The Journal of Systems Software*, 43:177-185, 1998.
- [78] Heckerman, D. "Probabilistic Interpretations for Mycin's Certainty Factors". *Uncertainty in Artificial Intelligence*. Págs: 167-196. LN. Kanal and J.F. Lemmer (Editors), 1986.
- [79] Heckerman, D. "A Tutorial on Learning With Bayesian Networks", Technical Report, MSR-TR-95-06, 1996. En: <ftp://ftp.research.microsoft.com/pub/dtg/david/tutorial.ps>
- [80] Henderson-Sellers, B. "Who needs an object-oriented methodology anyway". *JOOP*, 8(6):6-8. 1996.
- [81] Henderson-Sellers, B. "A methodological metamodel of process". *JOOP*, 11(9):56-58; 63, 1999.
- [82] Hess, M. "Mixed-level knowledge representations and variable-depth inference in natural language processing". *International Journal on Artificial Intelligence Tools (IJAIT)*, 6(4):481-509, 1997.

- [83] **Horvitz, E.O.; Heckerman, D.** "The inconsistent use of measures of certainty in Artificial Intelligence research", *Uncertainty in Artificial Intelligence*. LN. Kanal and J.F. Lemmer (Editors), Págs: 137-151, 1986.
- [84] **Hu, C.; Wang, F.** "Constructing an Integrated Visual Programming Environment". *Software-Practice and experience*, 28(7):773-798, 1998.
- [85] **Hurwitz, G.; Furth, J.J.** "Messenger RNA". *Scientific American*, 206(2):41-49, feb. 1962.
- [86] **Iivari, J.** "Object-oriented as structural, functional and behavioural modelling: a comparison of six methods for object-oriented analysis." *Information and Software Technology*, 37(3):155-163, 1995.
- [87] **Ishikawa, H.; Yamane, Y.; Izumida, Y.; Kawato, N.** "An Object-Oriented Database System Jasmine: Implementation, Application and Extension". *IEEE Transactions on Knowledge and Data Engineering*, 8(2):285-304, 1996.
- [88] **Jaaksi, A.** "Object Oriented Specification of User Interfaces", *Software-Practice and Experience*, 25(11):1203-1221, 1995.
- [89] **Jaaksi, A.** "A Method for your First-Oriented Project". *JOOP*, 10(8): 17-25, 1998.
- [90] **Jackson, M.** "System Development". Ed. Prentice Hall, 1983.
- [91] **Jacobson, I.** "Object-Oriented Software Engineering. A use case driven approach" Addison-Wesley. Reading. M.A. 1992.
- [92] **Jacobson, I.; Christerson, M.** "Modelling with the use cases: A growing consensus on use cases" *JOOP*, 8(1):15-19, 1995.
- [93] **Jensen, F.; Jensen, F.V.; Dittmer, S.L.** "From Influence Diagrams to Junction Trees" . (367-373) R. L. de Mántaras and D. Poole, editors, Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, Seattle, Washington, July 29-31. Morgan Kaufmann, San Mateo, California, 1994.
- [94] **Johnson, R.E.** "Documenting Frameworks using Patterns". *ACM SIGPLAN Notices*, 27(10), october, 1992.
- [95] **Kardell, M.** "A classification of object-oriented design patterns". Master's thesis, Umea university, 1997.
- [96] **Kernighan, B.; Ritchie, D.M.** "The C programming language" Englewood Cliffs, Nueva Jersey, Prentice-Hall, 1978.
- [97] **Kerth, N.** "Caterpillar's Fate: (metamorphosis) A Pattern Language for Transformation from Analysis to Design", Capítulo 16 del libro: *Pattern Languages of Program Design*, James O. Coplien, Douglas C. Schmidt, (Eds), 1994.
- [98] **Kim, J. H.; Pearl, J.** "A computational model for combined causal and diagnostic reasoning in inference systems" *Proceedings of the 8<sup>th</sup> International Joint Conference on Artificial Intelligence (IJCAI – 83)*. Págs: 190-193, 1983.
- [99] **Kiper, J. D.** "Structural Testing of Rule-Based Expert Systems". *ACM Transactions on Software Engineering and Methodology*, 1(2):168-187, 1992.
- [100] **Koestler, A.** "The Ghost in the Machine", Hutchinson/Danube, London, 1967 / 1976.
- [101] **Koller, D.; Pfeffer, A.** "Object Oriented Bayesian Networks" *Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI)*, Providence, Rhode Island, Págs: 302-313, 1997
- [102] **Kulikowski, C.A.; Weiss, S.M.** "Computer-based models of glaucoma". Department of Computer Science. Rutgers University. *Computers in Biomedicine Report* nº 3, 1971.
- [103] **Kulikowski, C.A.; Weiss, S.M** "Representation of expert knowledge for consultation: the CASNET and EXPERT projects". *Artificial Intelligence in Medicine*, Págs: 21-25. Westview Press, Szolovits ed. Boulder C.O., 1982.
- [104] **Kuntzmann-Combelle, A.; Vogel, C.** "KOD: A support environment for cognitive acquisition and management". *Safety and reliability society symposium*, 1988.
- [105] **Landauer, "Correctness principles for rule-based expert systems".** *Expert Systems With Applications*, 1(3):291-316, 1990.

- [106] **Lane, A.** "Object-Oriented Turbo Pascal". Prentice-Hall / M&T Books, ISBN 13-629916-4, Pbk; 13-629924-5, Pbk/Disk, 1990.
- [107] **Lauritzen, S.L.; Spiegelhalter, D.J.** "Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems". *Journal of the Royal Statistical Society B.*, 50(2):157-224, 1988.
- [108] **Lauritzen, S. L.** "Propagation of probabilities, means, and variances in mixed graphical models". *Journal of the American Statistical Association (Theory and Methods)*, 87(420):1098-1108, 1992.
- [109] **Lee, S.; O'Keefe, R.M.** "The effect of Knowledge Representation Schemes on Maintainability of Knowledge-Based Systems". *IEEE Transactions on knowledge and data engineering*, 8(1):173-178, 1996.
- [110] **Liebowitz, J.** "Introduction to Expert Systems". Santa Cruz, CA: Mitchell Publishing, Inc., 1988.
- [111] **Liu, S.; Jeff-Offutt, O.A.; Ho-Stuart C.; Sun, Y.; Ohba, M.** "SOFL: A Formal Engineering Methodology for Industrial Applications". *IEEE Transactions on Software Engineering*, 24(1):24-45, 1998.
- [112] **Losavio, F. ; Matteo, A.** "A Method for User-Interface Development". *JOOP*, 10(5):22-27, 1997.
- [113] **Mahajan, R.; Schneiderman, B.** "Visual and Textual Consistency Checking Tools for Graphical User Interfaces". *IEEE Transactions on Software Engineering*, 23(11):722-735, 1997.
- [114] **Maher Hakim, M.; Garret Jr, J. H.** "An object-centered approach for modelling engineering design products: Combining description logic and object-oriented modelling". *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 11:187-198, 1997.
- [115] **Mathews, J.** "Organisational Foundations of Object-Oriented Programming". *Journal Systems Software*, 34:247-253, 1996.
- [116] **McCarthy, J.** "Programs with Common Sense". Proceedings of the Teddington Conference on the Mechanization of Thought Processes, London: Her Majesty's Stationery Office, 1959.
- [117] **McCarthy, J.** "Recursive functions of symbolic expressions and their computation by machine" *Communications of the ACM*, 3:184-195, 1960.
- [118] **McCarthy, J.** "History of LISP" *SIGPLAN Notices*, 13:217-223, 1978.
- [119] **Meseguer, P.; Verdaguer, A;** "Verification of Multi-Level Rule-Based Expert Systems: Theory and Practice". *International Journal of Expert Systems*, 6(2):163-192, 1993.
- [120] **Meszaros, G; Doble, J.** "A Pattern Language for Pattern Writing". *Pattern-languages of program design-3*. Págs. 529-574, 1998.
- [121] **Minsky, M.** "A framework for representing knowledge". *The psychology of computer vision*, Winston, P. (Ed). Págs:211-277, Ed. McGraw-Hill, New York, 1974.
- [122] **Moore, R.C.** "The role of logic in knowledge representation and common-sense reasoning", Proc. of the AAAI-82, Pittsburg, PA, Págs: 428-433, 1982.
- [123] **Moseley, L.; Dobson, O.** "Knowledge representation for reasoning: Tables, frames and rules in a cutting fluids application", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 10:37-45, 1996.
- [124] **Muller, P.A.** "Modelado de Objetos con UML". Ed. Gestión 2000, S.A., Barcelona, 1997.
- [125] **Murrel, S; Plant, R.** "On the validation and verification of production systems: a graph reduction approach". *International Journal Human Computer Studies*, 44:127-144, 1996.
- [126] **Mylopoulos, J.; Wang, H.; Kramer, B.** "Knowbel: A Hybrid Expert System Building Tool and Its Applications". *IEEE Expert*, February, Págs: 17-24, 1993.
- [127] **Nazareth, D.L.; Kennedy, M.H.;** "Knowledge-Based System Verification, Validation and Testing: The Evolution of a Discipline". *International Journal of Expert Systems*, 6(2):143-162, 1993.

- [128] Newell, A.; Simon, H.A. "Human Problem Solving" Prentice-Hall, Englewood Cliffs, NJ, 1972.
- [129] Nguyen, T.A.; Perkins, W.A.; Laffety, T.J.; Pecora, D. "Checking an expert system knowledge base for consistency and completeness" *IJCAI-85*, 1:375-378, 1985.
- [130] Nygard, K. "Basic concepts in Object Oriented Programming". *SIGPLAN Notices*, 21(10), October, 1986.
- [131] Passioura, J.B. "Accountability, Philosophy and Plant Physiology" *Australian Journal of Science*, 10(10):347-350, 1979.
- [132] Pauker, S.G.; Gorry, G.A.; Kassirer, J.P.; Schwartz, W.B. "Towards the simulation of clinical cognition. Taking a present illness by computer". *American Journal of Medicine*, 60:981-996, 1976.
- [133] Pearl, J. "Fusion, Propagation, and Structuring in Belief Networks". *Artificial Intelligence*, 29:241-288, 1986.
- [134] Pearl, J. "Probabilistic Reasoning in Intelligent Systems". Morgan Kaufmann, San Mateo, California, 1988.
- [135] Pearl, J. "Bayesian networks". *Technical Report (R-246) Revision I, MIT Encyclopaedia of the Cognitive Sciences*, July, 1997.
- [136] Peot, M.A.; Shachter, R.D. "Fusion and Propagation with Multiple Observations in Belief Networks" *Artificial Intelligence*, 48:299-318, 1991.
- [137] Philip, G.C. "Software design guidelines for event-driven programming". *The Journal of Systems and Software*, 41:79-91, 1998.
- [138] Phillips, J.S.; Sanders, P. "First Steps in Prototyping". *AI EXPERT*, 3(5):64-68, 1988.
- [139] Piattini Velthuis, M.G. "Definición de una metodología de desarrollo para bases de datos orientadas al objeto fundamentadas en extensiones del modelo relacional". Tesis, Dpto. de lenguajes y sistemas informáticos e ingeniería del software, facultad de Informática, UPM, 1994.
- [140] Preece, A.D.; Shinghal, R.; Batarek, A. "Verifying Expert Systems: A Logical Framework and a Practical Tool". *Expert Systems With Applications*, 5:421-436, 1992.
- [141] Preece, A.D.; Grossner, C.; Radhakrishnan, T.; "Validating dynamic properties of rule-based systems". *International Journal Human Computer Studies*, 44:145-169, 1996.
- [142] Qiu, D. "Certainty Factor theory and its implementation in an information system-oriented expert system shell", *Applied Artificial Intelligence*, 6:147-166, 1992.
- [143] Quillian, R. "Semantic memory". *Semantic Information Processing*, Minsky, M. (Eds), Cambridge, Massachusetts, Institute of Technology Press, 1968.
- [144] Quinlan, J.R.; "Induction of decision trees", *Machine-Learning*, 1(1): 81-106, 1986.
- [145] Riehle, D.; Zullighoven, H. "Understanding and using patterns in software development". *Theory-and-Practice-of-Object-Systems*, 2(1):3-13, 1996.
- [146] Riley, G. "CLIPS: An Expert System Tool for Delivery and Training". *Proceedings of the Third Conference on Artificial Intelligence for Space Applications*, Huntsville, AL, 1987.
- [147] Rising, L. "Patterns: A Way to Reuse Expertise" *IEEE-Communications-Magazine*, 37(4):34-66, April, 1999.
- [148] Roberts, D.; Johnson, R. "Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks". University of Illinois.
- [149] Rumbaugh, J.; Blaha, M.; Premiérani, W.; Eddy, F.; Lorenzen, W. "Object-Oriented Modeling and Design", Prentice-Hall, Englewood Cliffs, NJ, 1991.
- [150] Rumbaugh, J. "Getting started: Using Use Cases to Capture Requirements", *JOOP*, Septiembre, Págs: 8-23, 1994.
- [151] Rumbaugh, J. "Modelling through the Years" *JOOP*, 10(4):16-19, 1997.

- [152] **Saffiotti, A.; Umkehrer, E.** "PULCINELLA. A general Tool for Propagating Uncertainty in Valuation Networks" *Procs. Of the 7<sup>th</sup> on Uncertainty in AI*. Los Angeles, CA 323-331, 1991. En: <http://iridia.ulb.ac.be/pulcinella>
- [153] **Salingaros, N.A.** "The Structure of Pattern Languages", to appear in *Architectural Research Quarterly, 2000*. En: <http://www.math.utsa.edu/phere/salingar/StructurePattern.html>
- [154] **Sarabhai, A.S.; Stretton, A.O.W.; Brenner, S.; Bolle, A.** "Co-linearity of the gene with the polypeptide chain". *Nature*, 201(4914):13-17, 1964.
- [155] **Schmidt, D.C.; Johnson, R.E. Fayad, M.** "Software Patterns" Guest Editorial for the *Communications of the ACM. Special Issue on Patterns and Pattern Languages*, 39(10), 1996.
- [156] **Schmidt, D.C.** "Wrapper Facade. A Structural Pattern for Encapsulating Functions within Clases", *C++ Report*, 11, February 1999.
- [157] **Shachter, R.D.; Andersen, S.K.; Szolovits, P.** "Global Conditioning for Probabilistic Inference in Belief Networks", *Proc. of the 10th. Conference on Uncertainty in Artificial Intelligence*, Págs: 514-522, 1994.
- [158] **Shafer, G.** "A Mathematical Theory of Evidence", Princeton University Press, Princeton, N.J., 1976.
- [159] **Shafer, G.** "Probability Judgement in Artificial Intelligence and Expert Systems", *Statistical Science*, 2:3-44, 1987.
- [160] **Shiu, S.C.K.; Liu, J.N.K.; Yeung, D.S.** "Formal Description and Verification of Hybrid Rule/Frame Based Expert Systems" *Expert Systems With Applications*, 13(3):215-230, 1997.
- [161] **Shortliffe, E.H.; Buchanan, B.G.** "A Model of Inexact Reasoning in Medicine". *Mathematical Biosciences*, 23:351-379, 1975.
- [162] **Shortliffe, E.H.** "Model of Inexact Reasoning in Medicine". *Artificial Intelligence Series 2. Compute Based Medical Consultations: MYCIN*, Capít. 4. Págs: 159-194, 1976.
- [163] **Shumate, K.** "Structured Analysis and Object Oriented Design are Compatible". *ACM ADA Letters*, 11(4):78-90, 1991.
- [164] **Smith, R.W.; Kieronska, D.; Venkatesh, S.** "Conceptual representation for multimedia information". *International Journal of Pattern Recognition and Artificial Intelligence*, 11(2):303-327, 1997.
- [165] **Spiegelhalter, D.J.; Lauritzen, S.L.** "Sequential Updating of Conditional Probabilities on Directed Graphical Structures", *Networks*, 20:579-605, 1990.
- [166] **Spiegelhalter, D. J.; Dawid, A. P.; Lauritzen, S. L.; Cowell, R. G.** "Bayesian Analysis in Expert Systems", *Statistical Science*, 8(3):219-283, 1993.
- [167] **Stroustrup, B.** "The C++ Programming Language" Addison-Wesley / Reading, Massachusetts, 1986.
- [168] **Suwa, M.; Scott, A. C.; Shortliffe, E. H.** "An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System". *AI Magazine*, Fall, Págs: 16-21, 1982
- [169] **Szolovits, P.; Pauker, S.G.** "Categorical and probabilistic reasoning in medicine" *Artificial Intelligence*, 11:115-144, 1978.
- [170] **Tessem, B.** "Approximations for efficient computation in the Theory of Evidence". *Artificial Intelligence*, 61(2):315-329, 1993.
- [171] **Turing, A.M.** "Computing machinery and intelligence". *Mind*, 59:433-460, 1950.
- [172] **Vadaparty, K.** "Bridging the gap between objects and tables: object and data models". *JOOP*, 12(2), 1999.
- [173] **Van Melle, W.; Shortliffe, E.H.; Buchanan, B.G.** "EMYCIN: A knowledge engineer's tool for constructing rule-based expert systems". *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*, B.G. Buchanan y E.H. Shortliffe (Eds), cap. 15. Págs. 302-313, Addison-Wesley, Reading, MA, 1984.
- [174] **Vlissides, J.** "Pattern Hatching. Seven Habits of successful Pattern Writers" *C++ Report*, November/December, 1996.

- [175] Voorbraak, F. "A computationally efficient approximation of Dempster-Shafer theory". *International Journal of Man Machine Studies*, 30(5):525-536, 1989.
- [176] Wagner, A. "A formal Object Specification technique integrating object and functional model". *International Journal of Software Engineering and Knowledge Engineering*, 7(4):503-524, 1997.
- [177] Walczak, S.. "Knowledge acquisition and knowledge representation with class: the object-oriented paradigm". *Expert Systems with Applications*, 15:235-244, 1998.
- [178] Walley, P.T. "Measures of uncertainty in expert systems". *Artificial Intelligence*, 83:1-58, 1996.
- [179] Ward, P.T.; Mellor, S.J. "Structured Development for real-time Systems". N.Y.: Yourdon Press, 1985.
- [180] Ward, P.T. "How to Integrate Object Orientation With Structured Analysis and Design". *IEEE Software*, 6:74-82, 1989.
- [181] Waterman, D.A. "A guide to Expert Systems". Readings, MA: Addison-Wesley, 1986.
- [182] Watson, J.D.; Crick, F.H.C. "Molecular structure of nucleic acids". *Nature*, 171(4356):737, Abril 25, 1953.
- [183] Weiss, S.M.; Kulikowski, C.A.; Amarel, S.; Safir, A. "A model-based method for computer-aided medical decision-making". *Artificial Intelligence*, 11:145-172, 1978.
- [184] Whitenack, Jr, B.G. "RAPPeL: A Requirements Analysis Process Pattern Language for Object Oriented Development" Capit 15 del libro: *Pattern Languages of Program Design*, James O. Coplien, Douglas C. Schmidt, Editors, 1994.
- [185] Wielinga, B.J.; Schreiber, A.Th; Breuker, A. "KADS: A modelling approach to knowledge engineering". *Knowledge Acquisition*, 4:5-53, 1992.
- [186] Wieringa, R. "A survey of structured and object-oriented software specification methods and techniques". *ACM Computing Surveys*, 30(4):459-527, 1998.
- [187] Wirfs-Brook, R.J.; Johnson, R.E. "Current Research in Object Oriented Design". *Communications of the ACM*, 33(9):104-124, 1990.
- [188] Wirth, N. "The Programming Language Pascal". *Acta Informatica*, 1:35-63, 1971.
- [189] Wolber, D. "Reviving Functional Decomposition in Object-Oriented Design". *JOOP*, 10(6):31-38, 1997.
- [190] Wu, X, "Explicit schematic information in knowledge representation and acquisition", *Expert Systems with Applications*, 15:215-221, 1998.
- [191] Wyatt, H.V. "When does Information become knowledge?". *Nature*, 235:86-89, January, 14, 1972.
- [192] Xiang, Y.; Pant, B.; Eisen, A.; Beddoes, M.P.; Poole. D. "Multiply Sectioned Bayesian Networks For Neuromuscular Diagnosis", *Artificial Intelligence in Medicine*, 5:293-314, 1993.
- [193] Xiang, Y. "Optimisation of Inter-Subnet Belief Updating in Multiply Sectioned Bayesian Networks". *Proc. Uncertainty in A.I. 95*. Págs: 563-573, Montreal, 1995.
- [194] Xiang, Y.; Wong, S.K.M.; Cercone, N. "Quantification of uncertainty in classification rules discovered from databases", *Computational Intelligence*, 11(2):425-441, 1995.
- [195] Xiaodong. Y.; Jiajun, C.; Guoliang, Z. "Two-dimensional Software Development Model Combining Object-Oriented Method with Formal Method". *Software Engineering Notes*, 23(1):81-85, 1998.
- [196] Xu, H. "An Efficient Implementation of Belief Function Propagation", Technical Report IRIDIA/91-4, 1991.
- [197] Yacoub, S.M.; Ammar, H. "Toward Pattern-Oriented Frameworks". *JOOP*, 12(8):25-35; 46, 2000.
- [198] Yen, John, "Implementing Evidential Reasoning in Expert Systems". *Uncertainty in Artificial Intelligence 3*, Elsevier Science Publishers, Inc., L. N. Kanal, T. S. Levitt, and J. F. Lemmer (Eds.), Págs: 333-344, 1989.

- [199] Yourdon, E.; Constantine, L.L. "Structured Design", Prentice Hall/Yourdon Press, 1978.
- [200] Zadeh, L.A. "Fuzzy Sets". *Information and Control*, 8:338-353, 1965.
- [201] Zadeh, Lofti A. "Fuzzy Sets as a basis for a Theory of Possibility", *Fuzzy Sets and Systems*, 1:3-28, 1978.
- [202] Zadeh, Lofti A. "Knowledge representation in fuzzy logic". *An Introduction to fuzzy logic applications in Intelligent systems*, edited by Ronald R. Yager & Lofti A. Zadeh, Kimmer Academic Publishers, Págs: 1-25, 1992.
- [203] Zendler, A.M. "Foundation of the taxonomic object system" *Information and Software Technology*, 40:475-492, 1998.
- [204] Zhang, D.; Nguyen, D. "PREPARE: A Tool for Knowledge Base Verification". *IEEE Transactions on Knowledge and Data Engineering*, 6(6):983-989, Diciembre, 1994.
- [205] Zlatareva, N.P. "A refinement framework to support validation and maintenance of knowledge-based systems" *Expert Systems with Applications*, 15:245-252, 1998.



**ABRIR TOMO II**

