CAPITULO 1

MÉTODOS ESTRUCTURADOS DE REPRESENTACIÓN DEL CONOCIMIENTO

- **Redes Semánticas**
- **Modelos de Dependencia Conceptual**
- **“Frames” y Guiones**
- **Paradigma de Orientación a Objetos**
- **Reglas de Producción**
- **Resumen**
- **Textos Básicos**

1. MÉTODOS ESTRUCTURADOS DE REPRESENTACIÓN DEL CONOCIMIENTO

La lógica formal permite la utilización de procedimientos de resolución que posibilitan el razonamiento con hechos. Sin embargo, los objetos del universo real tienen propiedades y se relacionan con otros objetos. Así, es útil disponer de estructuras de representación que permitan, por una parte, agrupar propiedades, y por otra, obtener descripciones únicas de objetos complejos.

En cualquier caso, los objetos no son las únicas entidades estructuradas del universo. También sería muy útil poder representar eficazmente escenarios y secuencias típicas de acontecimientos.

Para tratar de dar respuesta a tales cuestiones, en inteligencia artificial se utilizan *esquemas no formales de representación del conocimiento*. Tales esquemas son fundamentalmente métodos estructurados de representación, y tienen que verificar las siguientes propiedades:

ADECUACIÓN REPRESENTACIONAL:

> El esquema elegido debe ser capaz de representar las distintas clases de conocimiento del dominio.

ADECUACIÓN INFERENCIAL:

> El esquema elegido debe permitir la manipulación del conocimiento para obtener conocimiento nuevo.

EFICIENCIA INFERENCIAL:

> El esquema elegido debe ser versátil utilizando aquella información que permita optimizar el proceso inferencial.

EFICACIA ADQUISICIONAL:

> El esquema elegido debe suministrar vías que permitan la incorporación de información y conocimiento nuevos.

Cualquier esquema estructurado de representación del conocimiento va a poder ser clasificado en una de las siguientes categorías:

métodos declarativos
métodos procedimentales

En los esquemas declarativos el conocimiento se representa como una colección estática de hechos para cuya manipulación se define un conjunto genérico y restringido de procedimientos. Los esquemas de este tipo presentan ciertas ventajas. Así, las verdades del dominio se almacenan una sola vez. Además, es fácil incrementar e incorporar nuevo conocimiento sin modificar ni alterar el ya existente.

En los esquemas procedimentales la mayor parte del conocimiento se representa como “procedimientos”, lo cual le confiere al esquema de representación un carácter dinámico. También los esquemas procedimentales presentan ventajas. Así:

Al dar prioridad a los procedimientos hacemos mayor énfasis en las capacidades inferenciales del sistema.

Permiten explorar distintos modelos y técnicas de razonamiento.

Permiten trabajar con falta de información y con datos de carácter probabilístico.

Incorporan de forma natural conocimiento de tipo heurístico.

Independientemente del esquema de representación elegido, es muy útil considerar una serie de elementos que nos permiten establecer relaciones entre distintas estructuras de conocimiento. Tales elementos son:

ES_UN (IS_A): que permite establecer relaciones entre taxonomías jerárquicas (Figura 1.1)

ES_PARTE_DE (PART_OF): que permite establecer relaciones entre objetos y componentes de un objeto (Figura 1.2)

Una propiedad importante de ambas relaciones es la transitividad. Veamos los siguientes ejemplos:

MILU ES UN PERRO	LA NARIZ ES PARTE DE LA CARA
UN PERRO ES UN ANIMAL	LA CARA ES PARTE DE LA CABEZA
MILU ES UN ANIMAL	LA NARIZ ES PARTE DE LA CABEZA

La transitividad está vinculada al razonamiento deductivo. Más importante aún es el hecho de que la transitividad de la relación ES_UN nos permite establecer un método para la obtención de propiedades de los objetos relacionados.

Ello configura un proceso de *herencia de propiedades* mediante el cual, si un objeto pertenece a una determinada clase, a través de la relación ES_UN dicho objeto hereda las propiedades de la clase. Desde una perspectiva formal podemos definir una correspondencia entre ambas relaciones y la lógica de predicados. Así, por ejemplo: hombre (MARCO), se convierte en ES_UN (MARCO, HOMBRE).

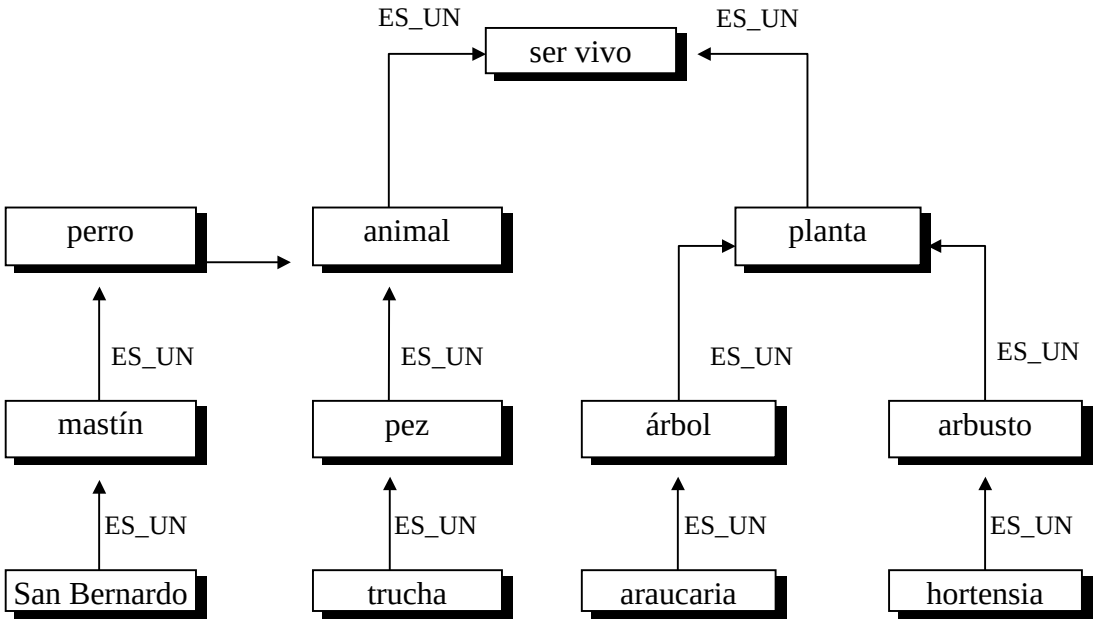


Figura 1.1 Distintos objetos conectados a través de relaciones ES_UN

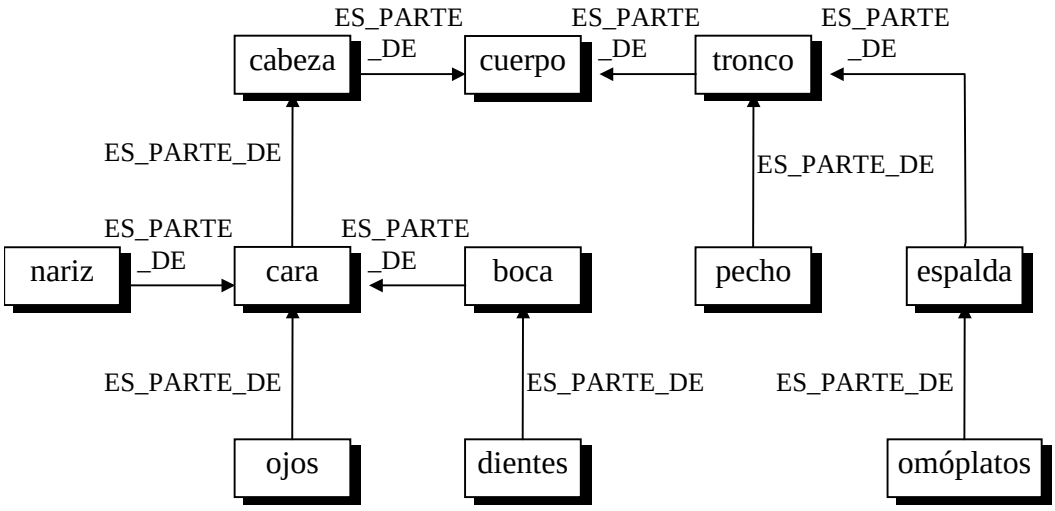


Figura 1.2 Objetos conectados a través de relaciones ES_PARTE_DE

Los métodos declarativos que estudiaremos en este tema serán:

REDES SEMANTICAS: Que permiten describir simultáneamente acontecimientos y objetos.

MODELOS DE DEPENDENCIA CONCEPTUAL: Estructuras especializadas que proporcionan mecanismos para representar relaciones entre los componentes de una acción.

FRAMES: Estructuras genéricas que permiten representar objetos complejos desde diferentes puntos de vista.

GUIONES: Estructuras especializadas, que derivan de las frames, y que son útiles para representar secuencias comunes de acontecimientos.

Aunque básicamente diferentes, estas cuatro estructuras de representación comparten ciertos elementos. Así, en todos ellos las entidades complejas pueden describirse como una colección de atributos y unos valores asociados (i.e., estructuras *slot-and-filler*).

También haremos, en este capítulo una breve descripción del paradigma de *orientación a objetos*, como un esquema alternativo y potente de representación jerárquica de objetos y clases, que comparten propiedades.

El método procedimental que estudiaremos será el de las *REGLAS DE PRODUCCION* (o simplemente REGLAS), sobre el cual haremos una extensión en otro capítulo a los llamados *SISTEMAS DE PRODUCCION*.

1.1. Redes Semánticas

Las redes semánticas son estructuras declarativas de representación en las que el conocimiento se representa como un conjunto de *nodos* conectados entre sí por medio de *arcos* etiquetados. Los arcos representan relaciones lingüísticas entre nodos (Figura 1.3)

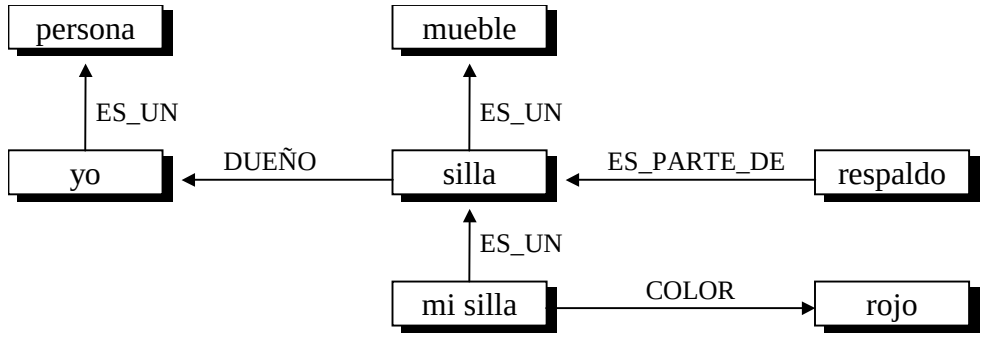


Figura 1.3 Una red semántica sencilla

En las redes semánticas cada propiedad incluye un enlace unidireccional. Para establecer enlaces bidireccionales hay que tratar cada relación por separado (Figura 1.4)

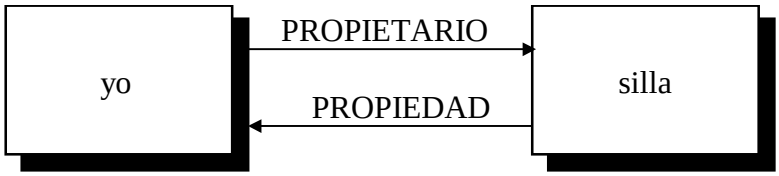


Figura 1.4 Un enlace bidireccional en una red semántica

Un enlace puede ser considerado como algo que aseveramos de un nodo en relación a otro. Puesto que una aseveración dada sólo puede ser *cierta* o *falsa*, un enlace es una relación binaria entre nodos.

Dos de las relaciones binarias más corrientes en las redes semánticas son las relaciones ES_UN y ES_PARTE_DE.

En este contexto, la relación ES_UN se emplea para establecer el hecho de que un elemento dado es miembro de una clase de elementos que tienen en común un conjunto de propiedades distinguibles.

Los conceptos de *clase* y de enlace ES_UN pueden emplearse también para representar situaciones, acciones y eventos.

Las relaciones más frecuentes en redes semánticas pueden clasificarse en una de las siguientes categorías:

OCURRENCIA:

Cuando se relaciona un miembro de una categoría general con la categoría a la que pertenece (se suele etiquetar “pertenece”).

GENERALIZACIÓN:

Cuando se relaciona una entidad con otra de carácter más general (ES_UN).

AGREGACIÓN:

Cuando se relacionan componentes de un objeto con el objeto propiamente dicho (ES_PARTE_DE).

ACCIÓN:

Cuando se establecen vínculos dinámicos entre diferentes objetos.

PROPIEDADES:

Que son relaciones entre objetos y características de los objetos.

La Figura 1.5 representa, por medio de una red semántica, la siguiente frase compleja: “María (sujeto de la acción), le dio (pretérito indefinido del verbo DAR), a Juan (beneficiario de la acción), los *principios de inteligencia artificial* (objeto directo), que es un libro de color azul que pertenece a la biblioteca de la Facultad de Informática.”

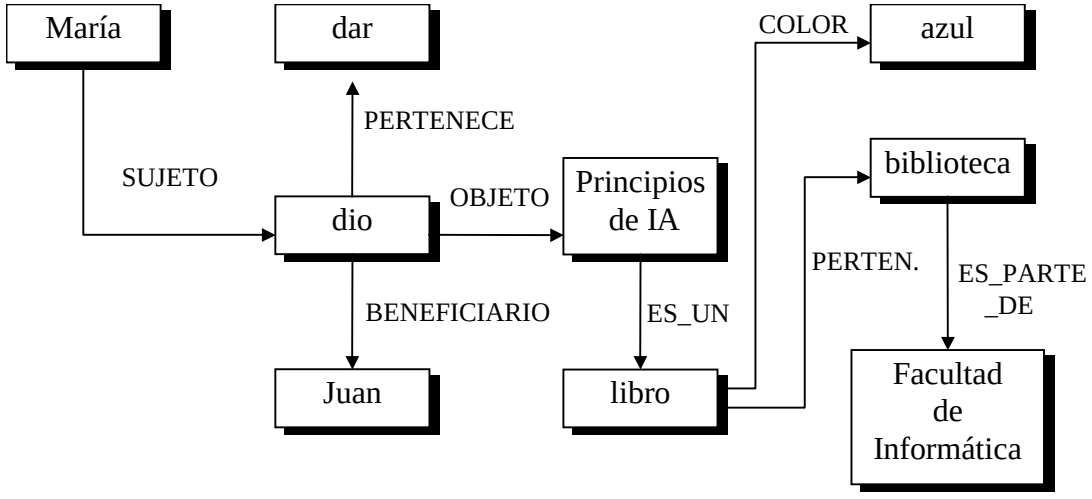


Figura 1.5 Una frase compleja representada a través de una red semántica sencilla

Desde una perspectiva computacional la implementación de una red semántica requiere la construcción de una tabla de *n* tuplas del tipo: OBJETO-ATRIBUTO-VALOR, de forma que:

El nodo padre sea el OBJETO
El arco sea el ATRIBUTO
El nodo destino sea el VALOR

La Figura 1.6 muestra un ejemplo de construcción de la tabla de tuplas asociada a una red semántica.

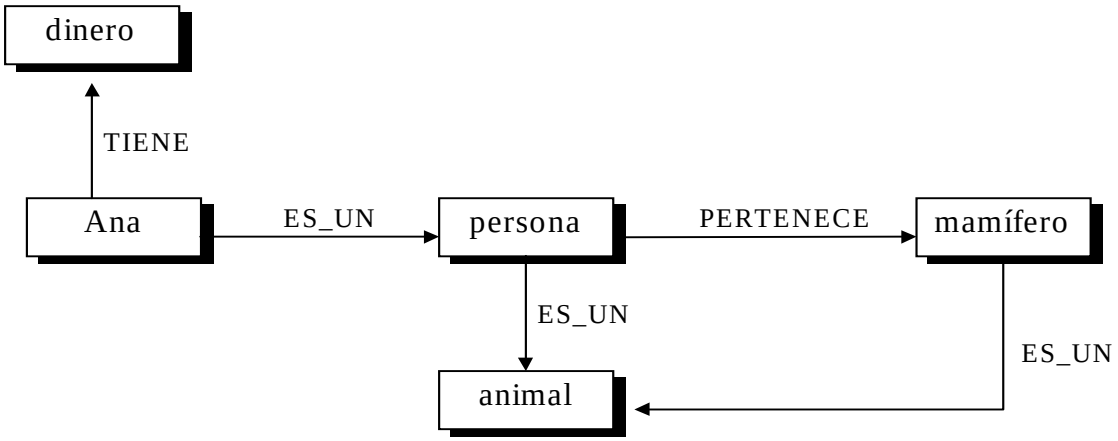


Figura 1.6 Un ejemplo de red semántica sencilla

Existe una correspondencia directa entre las redes semánticas y la lógica formal. Así, si utilizamos predicados, la representación de la red semántica de la figura 4.6 se haría por medio de los siguientes predicados:

TIENE (ANA, DINERO)
ES_UN (ANA, PERSONA)
ES_UN (PERSONA, ANIMAL)
PERTENECE (PERSONA, MAMIFERO)
ES_UN (MAMIFERO, ANIMAL)

La Tabla 1.1 muestra la tabla de tuplas asociada a la red semántica de la Figura 1.6.

OBJETO	ATRIBUTO	VALOR
Ana	tiene	dinero
Ana	es_un	persona
persona	es_un	animal
persona	pertenece	mamífero
mamífero	es_un	animal

Tabla 1.1 Tabla de tuplas asociada a una red semántica

Por otra parte, la representación interna del conocimiento de la red semántica de la Figura 1.6 podría ser la siguiente:

```
(ANA
        (TIENE (DINERO))
        (ES_UN (PERSONA))
)

(PERSONA
        (ES_UN (ANIMAL))
                (PERTENECE (MAMIFERO))
)

(MAMIFERO (ES_UN (ANIMAL)))
```

En redes semánticas, la herencia de propiedades nos dice que cualquier propiedad que consideremos cierta para una clase de elementos debe ser cierta para cualquier ejemplo de la clase. Este concepto hace que las redes semánticas sean particularmente interesantes para representar dominios que se puedan estructurar como taxonomías.

En cuanto a la forma de razonar con redes semánticas, el modelo permite obtener asociaciones simplemente rastreando los enlaces del sistema. Así, en la siguiente red semántica sencilla:

12 “mayor que” 7 “mayor que” 3

el rastreo nos permite concluir que “12 es mayor que 3”. No obstante, ninguna regla semántica rigurosa guía el proceso. En un sistema de lógica formal, las inferencias se realizan sobre la base de manejos sintácticos uniformes de símbolos y, por lo tanto, son siempre válidas (aunque en ocasiones puedan ser irrelevantes). Por el contrario, en redes semánticas las relaciones que se representan pueden no ser totalmente rigurosas (la mayoría de las veces por condiciones de excepción no reconocidas) y, por lo tanto, las inferencias obtenidas por rastreo pueden no ser válidas. Por ejemplo, la red de la Figura 1.7 es esencialmente cierta. Aplicando sobre esta red un proceso inferencial de rastreo, podríamos concluir que *Furia* tiene rabo. Sin embargo, existe una variedad de caballos cuyos ejemplares nacen sin rabo -caballos “mochos”- (i.e., condición de excepción no reconocida). Evidentemente, la inferencia obtenida sería falsa si *Furia* fuese realmente un caballo mocho.

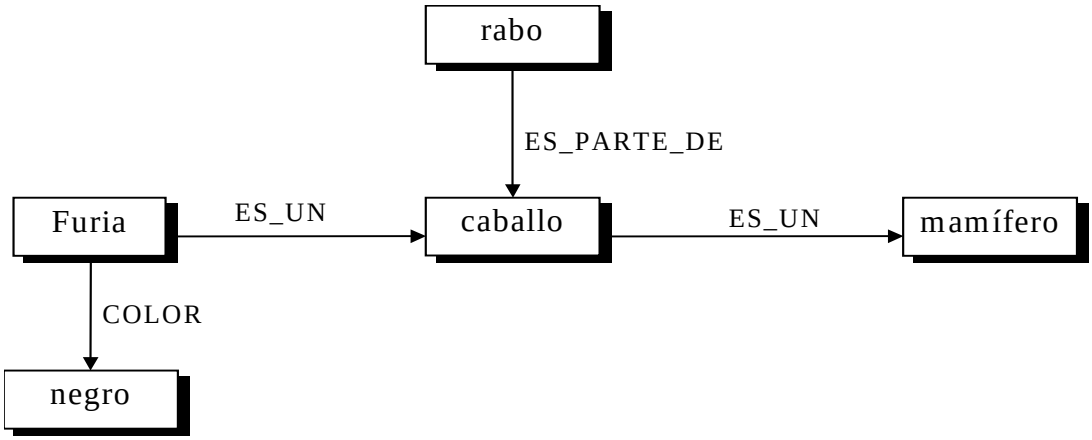


Figura 1.7 Red semántica “esencialmente cierta”

Otra técnica corriente de razonamiento con redes semánticas es el *emparejamiento*, consistente en la construcción de fragmentos de red, algunos de cuyos nodos tienen valores definidos, pero otros no. En este caso los nodos sin valores se representan por variables. El sistema debe entonces tratar de encontrar un fragmento de la red semántica original que encaje perfectamente en el fragmento de red semántica que hemos construido para representar nuestro problema.

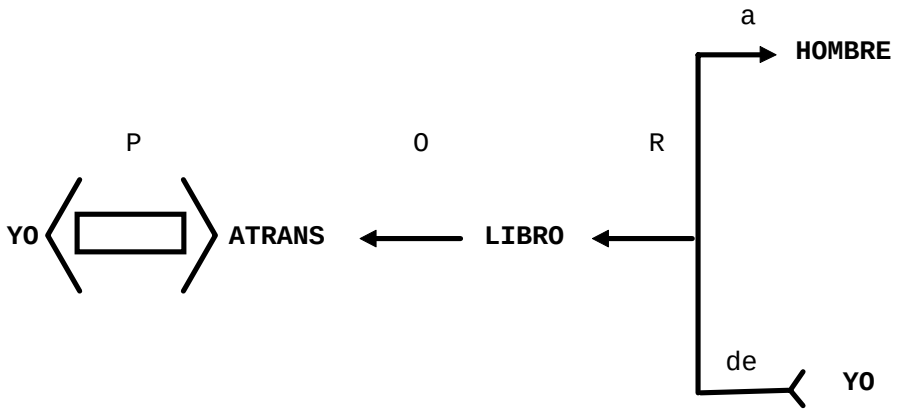
1.2. Modelos de Dependencia Conceptual

La dependencia conceptual intenta representar el significado de frases en lenguaje natural de forma que:

Se posibilite la derivación de conclusiones.
La representación sea independiente del lenguaje utilizado en las declaraciones originales.

Una representación de dependencia conceptual no se construye a partir de primitivas que correspondan a palabras concretas de una declaración dada. Por el contrario, se utilizan primitivas conceptuales que pueden combinarse para formar significados de palabras en cualquier lenguaje.

La dependencia conceptual proporciona simultáneamente una estructura y un conjunto específico de primitivas a partir de las cuales pueden construirse representaciones de elementos concretos de información. Así, el suceso representado por la frase: “Yo le di un libro al hombre”, se representaría en un modelo de dependencia conceptual del siguiente modo:



en donde:

Las flechas indican una dirección de dependencia
La flecha doble indica un enlace bidireccional entre la acción y el actor
"p" indica tiempo pasado
ATRANS es una de las primitivas de acción permitidas por el modelo e indica una transferencia de posesión
"o" indica una relación causal con un objeto
"R" indica una relación causal relativa a "libro", que va desde *Yo* hasta *hombre*

El conjunto de primitivas que utiliza el modelo es el siguiente:

ATRANS:	Que indica transferencia de una relación abstracta (e.g., dar)
FTRANS:	Que indica transferencia de la localización física de un objeto (e.g. ir)
IMPELER:	Que indica aplicación de fuerza física a un objeto (e.g., empujar)
MOVER:	Que indica movimiento de parte de un cuerpo por su propietario (e.g., patalear)
ASIR:	Que indica el hecho de tomar un actor un objeto (e.g., lanzar)
INGER:	Que indica ingerir un animal un objeto (e.g., comer)
EXPEL:	Que indica expulsar algo del cuerpo de un animal (e.g., llorar)
MTRANS:	Que indica transferencia de información mental (e.g., contar)
MCONST:	Que indica construir información nueva a partir de otra ya existente (e.g., decidir)
HABLAR:	Que indica producción de sonidos (e.g., decir)
ATENT:	Que indica centrar un órgano sensorial sobre un estímulo (e.g., oir)

Un segundo bloque de construcción es el conjunto de las dependencias permitidas entre las conceptualizaciones descritas en una frase. El modelo de dependencia conceptual incluye cuatro categorías conceptuales primitivas, a partir de cuyas dependencias pueden construirse las estructuras:

ACC Acciones

OB Objetos (que generan imágenes)

MA Modificadores de acciones (ayudas a las acciones)

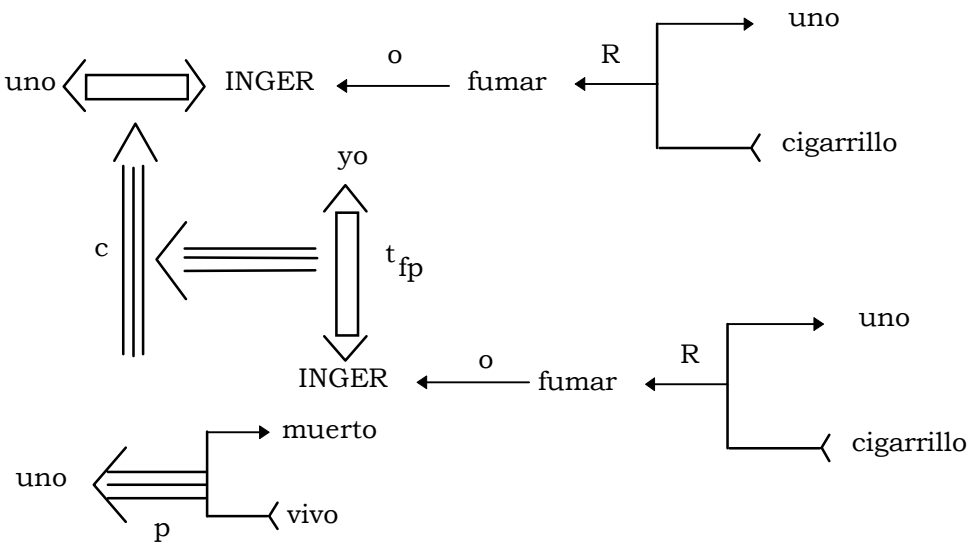
MO Modificadores de objetos (ayudas a las imágenes)

En este modelo, las dependencias entre conceptualizaciones corresponden a las relaciones semánticas entre conceptos subyacentes. Tales dependencias pueden resumirse en una lista de 15 reglas que no detallamos por no ser objetivo prioritario de este texto. El lector interesado podrá encontrar abundante información sobre este esquema de representación de conocimiento en la bibliografía que se incluye al final del libro.

Las conceptualizaciones representan sucesos que pueden modificarse de varias formas para proporcionar información indicada normalmente en el lenguaje por medio de: tiempo, modo y forma verbal. El conjunto de tiempos conceptuales utilizado habitualmente es el siguiente:

p pasado
f futuro
t transición
ti inicio de transición
tf final de transición
k continuar
? interrogación
/ negativo
nada presente
delta atemporal
c condicional

El uso de tiempos conceptuales es importante en el modelo de dependencia conceptual. A continuación se representa la frase "*puesto que el fumar puede matarte, lo dejé*", en donde se ilustra cómo este modelo emplea los tiempos conceptuales:



El modelo de dependencia conceptual facilita el razonamiento con el conocimiento representado ya que:

(1) Al descomponer el conocimiento en primitivas se necesitan menos reglas de inferencia.

(2) Muchas inferencias están autocontenidas en la representación.

(3) La estructura inicial construida con la información disponible, facilita el enfoque de atención del programa que debe entender las frases, ya que contiene huecos que deberán rellenarse.

1.3. "Frames" y Guiones

Ante un problema nuevo nadie empieza directamente un análisis exhaustivo, y desde cero, para construir incrementalmente estructuras de conocimiento cada vez más complejas, la última de las cuales describa perfectamente la nueva situación presentada. Por el contrario, el primer paso suele consistir en recuperar experiencias anteriores y tratar de razonar por "semejanza". El ejemplo es claro: *Este problema nuevo se parece a aquel otro que resolví de esta manera*.

Frames

Las *frames*, que pueden describirse como redes semánticas complejas que tratan el problema de la representación desde la óptica del razonamiento por semejanza. Describen clases de objetos y pueden definirse como representaciones estructuradas de conocimiento estereotipado.

Estructuralmente, una frame consta de una *cabecera*, que le da nombre a la frame (y que es representativa de la clase de objetos que se describen), y de un conjunto de *slots*, cada uno de los cuales representa una propiedad o atributo del elemento genérico representado por la frame. Cada slot puede tener distintos slots anidados y sin limitación de profundidad. De esta forma se habilitan posiciones concretas para ubicar sistemáticamente los componentes de nuestras experiencias anteriores en relación a la clase de elementos representados. Veamos un ejemplo:

AUTOMOVIL
    TIPOS
        TODO_TERRENO
        DEPORTIVO
        UTILITARIO
    COMPONENTES
        CARROCERIA
        PUERTAS
        CAPO
        ...
    ...

Cada una de las indentaciones de los slots representa un nivel de conocimiento o nivel epistemológico, y su contenido es una especialización del nivel anterior.

Un sistema en el que el conocimiento esté representado por medio de frames utiliza con profusión el concepto de herencia. Para ello se emplean slots de tipo ES_UN, que permiten la entrada de información a una frame, en un nivel de conocimiento determinado, y a partir del cual la información de la clase correspondiente pasa al objeto considerado. En este sentido, una frame puede ser un ejemplo de una frame de orden superior (ver Figura 1.8).

Las frames suelen incorporar también información procedimental. Para ello, ciertos slots llevan asociados procedimientos que la mayor parte del tiempo están inactivos, pero que cuando son activados desencadenan acciones concretas. Algunos de tales procedimientos, denominados *demons*, se muestran a continuación:

IF_NEEDED →
IF_ADDED →
IF_REMOVED → ... D_"Algún_Procedimiento"
IF_STORED →
IF_RETRIEVED →

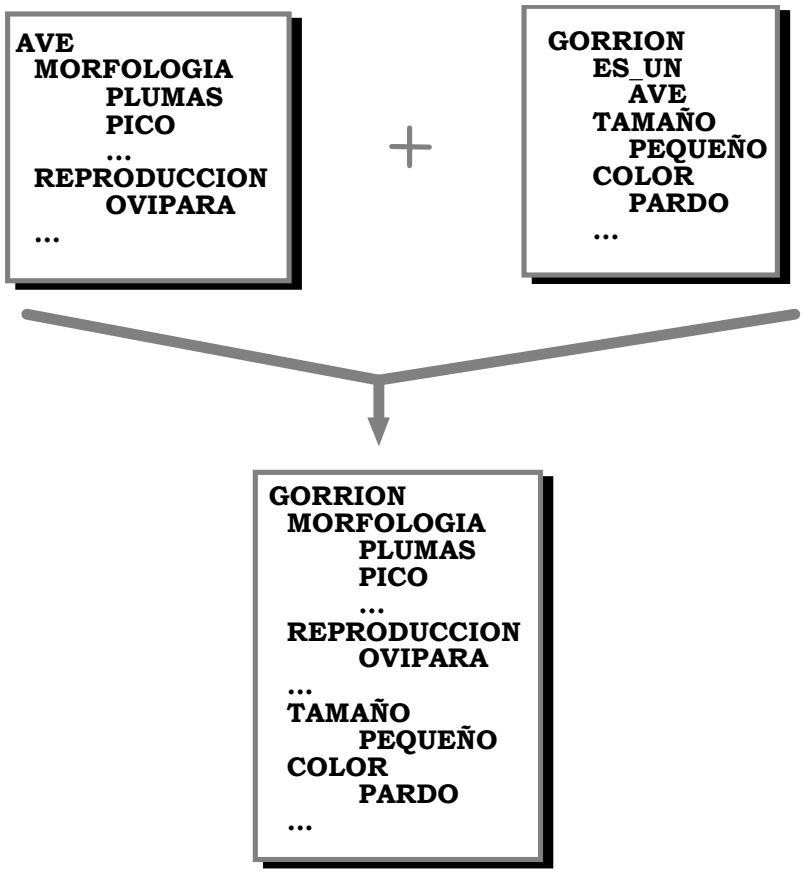


Figura 1.8 Frames y Herencia

Cuando un demon es activado (por una entrada en la frame, al nivel correspondiente), el procedimiento que sigue al demon es ejecutado y, normalmente, el demon (cumplida ya su misión), es eliminado inmediatamente (ver figura 4.9)

Los demons son estructuras muy útiles ya que:

Proporcionan uniones procedimentales entre distintas frames
Posibilitan la ejecución de rutinas externas
Imprimen un cierto carácter dinámico a la representación del conocimiento con frames

Como estructuras de representación, las frames pueden emplearse como:

Elementos descriptivos
Elementos de control del conocimiento

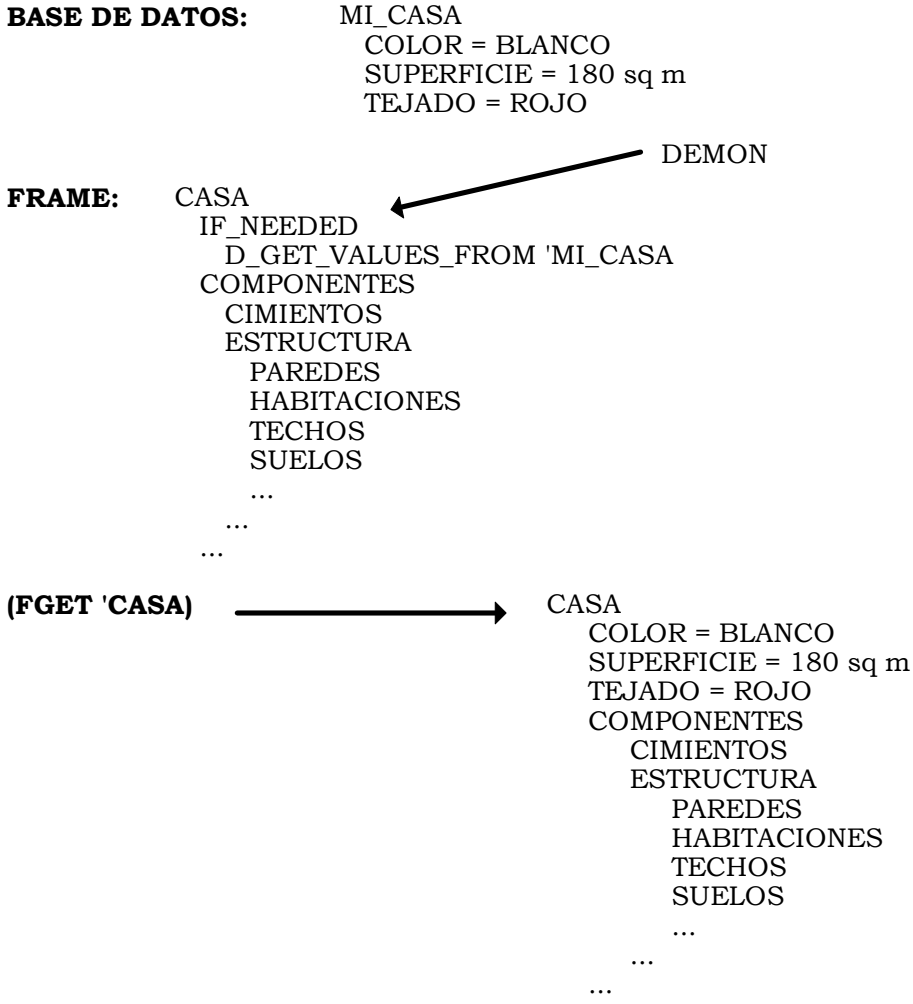


Figura 1.9 Ejemplo de cómo puede operar un 'demon'

La Figura 1.10 muestra una frame de carácter descriptivo, mientras que la Figura 1.11 muestra una frame diseñada para controlar el formato de entrada de un cierto parámetro que puede, a su vez, ser una frame o un slot de una frame.

Con lo visto, podemos ya resaltar que:

Las frames permiten definir procesos de razonamiento con información incompleta.

Las frames permiten inferir rápidamente hechos no representados de forma explícita.

Las frames imprimen cierto carácter dinámico a la representación al definir procesos que establecen relaciones entre otras frames y conexiones con el mundo exterior.

Las frames utilizan con profusión el concepto de herencia.

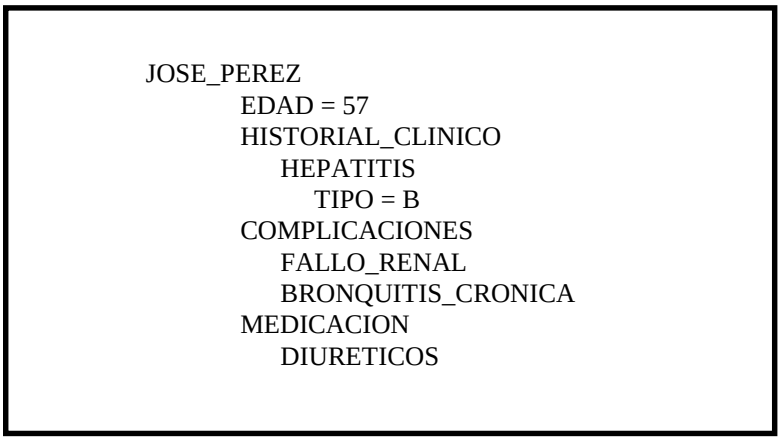


Figura 1.10 Una frame como elemento descriptivo

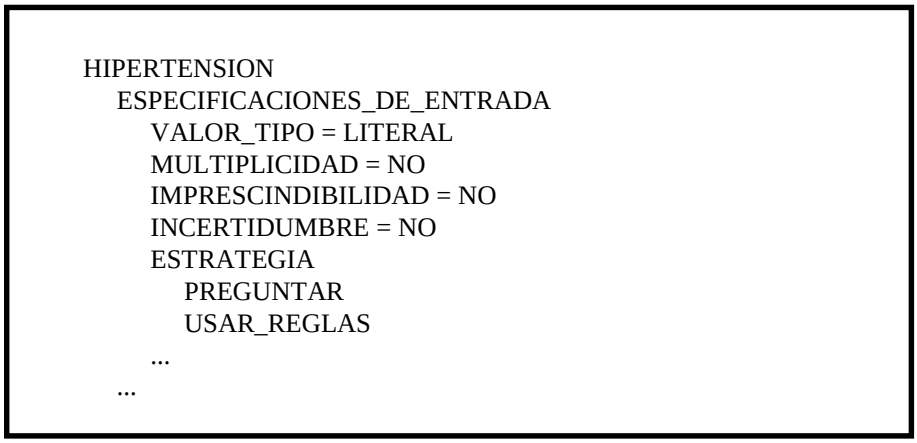


Figura 1.11 Una frame como elemento de control

El razonamiento con frames suele comenzar con la selección de una frame determinada que se ajuste a nuestra situación actual. Dado que en la mayoría de los casos no tendremos ninguna frame que describa exactamente nuestro estado inicial, comenzaremos seleccionando la más ajustada en base a la evidencia parcial disponible. A continuación, se produce la *ejemplificación* de la frame seleccionada tras considerar ciertas condiciones específicas actuales.

En general, el proceso de ejemplificación asocia un individuo particular con una clase. En otras palabras, obtenemos una descripción individual del problema actual considerando, por un lado, la descripción de la clase genérica y, por otro lado, las características específicas actuales.

Este procedimiento, aunque potente, está siempre sujeto y matizado por la experiencia individual. Así, en ocasiones, la subjetividad en la percepción desvirtúa la calidad de la representación.

Guiones

Los guiones son especializaciones del concepto general de frame, que conforman estructuras capaces de representar prototipos de secuencias de sucesos. En los guiones el tiempo es siempre una variable implícita. Los elementos estructurales más habituales de los guiones son:

Las condiciones de entrada o condiciones iniciales de aplicabilidad del guión.
Los resultados o hechos que serán verdaderos una vez el guión sea ejecutado.
Las herramientas u objetos relevantes en el desarrollo del guión.
Los papeles o roles que representan los actores que actúan en el guión.
Las escenas o secuencias típicas de eventos del guión.

En los guiones no existen reglas absolutas que definan contenidos genéricos. Así, un mismo evento puede ser contemplado según:

distintos factores de tiempo
distintos lugares de ocurrencia
distintos puntos de vista

La Figura 1.12 muestra el guión típico "viaje al teatro":

GUION: *Viaje al teatro*	ESCENA PRIMERA: Arranque
Herramientas:	Propietario busca llaves
Automóvil	Propietario abre puerta
Llaves	Propietario arranca coche
Puerta	Propietario sale
Aparcamiento	
	ESCENA SEGUNDA: Conducción
Papeles (roles):	
Propietario	Incorporación al tráfico
Conserje	Dirección al teatro
Entradas:	ESCENA TERCERA: Encuentro con el conserje
Propietario y automóvil	
en el punto de partida	Propietario para automóvil
	Propietario sale del coche
Resultado:	Propietario da llaves a conserje
Propietario en el teatro	Propietario entra al teatro
y automóvil en el	
aparcamiento	ESCENA CUARTA: Fin del guión
	Empleado entra al coche
	Empleado arranca
	Empleado encuentra aparcamiento libre
	Empleado aparca el coche
	Empleado sale del coche

Figura 1.12 El conocido guión de "Viaje al Teatro"

En los guiones el proceso de razonamiento está basado en que, si un determinado guión es apropiado para describir una situación dada, debe poder ser también utilizado para predecir acontecimientos no mencionados explícitamente. Así, para razonar sobre la base de un guión, éste debe ser previamente seleccionado mediante lo que se denomina proceso de *activación*.

Existen dos tipos fundamentales de guiones:

Los guiones instantáneos (i.e., aquellos a los que nos podemos referir siempre pero que no son el foco de atención principal de nuestro problema)

Los guiones no instantáneos (que constituyen el foco de atención principal de nuestro problema).

La activación de un guión instantáneo se realiza definiendo punteros en lugares estratégicos de forma que, cada vez que tal guión sea requerido, sepamos en dónde buscarlo.

Por el contrario, en el caso de los guiones no instantáneos procede la activación total del mismo tras el proceso de emparejamiento de la situación inicial con el conjunto de guiones que describen secuencias en nuestro dominio. Así, la secuencia de eventos en un guión puede entenderse como una *cadena causal* gigante.

1.4. Paradigma de Orientación a Objetos

La historia de la programación orientada a objetos comienza con el desarrollo del lenguaje de simulación de sucesos discretos – Simula – en Noruega en 1967, y continúa con el desarrollo de un lenguaje que casi toma como fetiche el concepto de objeto, Smalltalk, en los años 70. Desde mediados de la década de los 70 se produjo una fertilización recíproca entre la programación orientada a objetos y la investigación y el desarrollo en inteligencia artificial, lo que condujo a diversas y útiles extensiones de los lenguajes de IA, como LISP, y de los entornos de IA, como KEE o ART.

Los objetos aparecen como una extensión al concepto de frames y su principal característica es que encapsulan, en una misma representación, los datos y las estructuras procedimentales encargadas de manejar esos datos (recordemos que en las frames la información procedimental sólo aparecía en forma de demons).

Aunque esta aproximación pueda parecer poco natural, ya que los sistemas de inteligencia artificial siempre han tendido a separar la información declarativa de la información procedimental, se corresponde con una visión en la cual el mundo no está formado por datos y procedimientos sino por entidades que encapsulan sus propios datos y los procedimientos para manejarlos, lo que provee de una base más firme para el desarrollo de sistemas.

Un objeto se define como una colección de información (denominados normalmente atributos) que representa a una entidad del mundo real y una descripción de cómo esta información es manipulada (métodos). Por ejemplo un coche puede tener como atributos la matrícula, la marca, el precio y el año de compra. Además puede incluir métodos que calculen el precio actual del coche o el importe de su seguro. La notación gráfica de este objeto, utilizando el lenguaje de modelado unificado (UML), sería la que se muestra en la Figura 1.13:

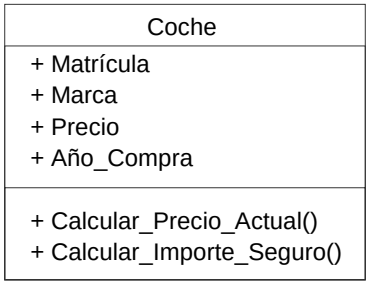


Figura 1.13 Ejemplo de clase con sus atributos y métodos

La forma de comunicación entre objetos es mediante el envío de mensajes entre los mismos. Mensajes y métodos son dos caras de la misma moneda, los métodos son los procedimientos que se invocan cuando un objeto recibe un mensaje. En terminología de programación tradicional, un mensaje es una llamada a una función. Los objetos pertenecen a clases, en la práctica una clase es como un esquema o plantilla que se utiliza para definir o crear objetos.

Según Booch los cuatro elementos más importantes del esquema de orientación a objetos son: abstracción, encapsulamiento, modularidad y jerarquía. Además de estos cuatro se suele añadir un quinto, polimorfismo, que ha cobrado tanta importancia que hoy en día no tiene sentido considerar un modelo de objetos que no soporte el polimorfismo. A continuación describiremos brevemente estas características.

Abstracción

La abstracción consiste en ignorar aspectos de una entidad que no son relevantes para el problema actual, de forma que podamos centrar nuestra atención en aquellos aspectos que sí lo son. La aproximación de la orientación a objetos fomenta que el desarrollador use abstracciones en los datos y en los procedimientos para simplificar su descripción del problema.

Definir una abstracción significa describir una entidad del mundo real, no importa lo compleja que pueda ser y, a continuación, utilizar esta descripción en un programa. El elemento clave de la abstracción es la clase. Una clase se puede definir como una descripción abstracta de un grupo de objetos, cada uno de los cuales se diferencia por su estado específico y por la posibilidad de realizar una serie de operaciones. Por ejemplo, una pluma estilográfica es un objeto que tiene un estado (llena de tinta o vacía) y sobre la cual se puede realizar algunas operaciones (por ejemplo, escribir, poner o quitar el capuchón, llenar de tinta si está vacía).

Encapsulamiento

El encapsulamiento se define como el proceso de almacenar en un mismo compartimiento los elementos de una abstracción que constituyen su estructura y su comportamiento. La abstracción y el encapsulamiento son conceptos complementarios: la abstracción se centra en el comportamiento observable de un objeto, mientras que el encapsulamiento se centra en la implementación que da lugar a ese comportamiento.

El encapsulamiento también implica ocultación de información, de forma que cada objeto revela lo menos posible de su estructura interna. Esta parte pública del

objeto es lo que se conoce como *interfaz*, mientras que los detalles internos del objeto que se ocultan al exterior se denominan *implementación*.

Así una operación (por ejemplo, la ordenación) es vista por sus usuarios como si fuera una simple entidad, aunque en realidad está formada por una secuencia de operaciones a bajo nivel. También un objeto (por ejemplo un coche) es visto como un simple objeto en vez de como una composición de sus partes individuales. La supresión de los detalles de bajo nivel nos permite razonar acerca de la operación u objeto de forma más eficiente.

Un encapsulamiento inteligente permite que cambios en el diseño afecten al sistema sólo de forma local. Así a medida que evoluciona un sistema, sus desarrolladores pueden ver que una aplicación real o bien ciertas operaciones llevan demasiado tiempo; o bien ciertos objetos consumen un espacio excesivo. En estas situaciones se suele cambiar la representación de un objeto, pero sin alterar su interfaz. Esta capacidad para cambiar la representación de una abstracción sin alterar a ninguno de los clientes que la utilizan es una de las ventajas más importantes del encapsulamiento.

Modularidad

El hecho de fragmentar un programa en componentes individuales suele contribuir a reducir su complejidad en algún grado, además de crear una serie de fronteras bien definidas y documentadas dentro del programa, que tienen un valor muy alto de cara a la comprensión del mismo.

La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de partes o módulos que sean cohesivos (agrupando abstracciones que guarden cierta relación lógica), y débilmente acoplados (minimizando las dependencias entre módulos).

Los principios de abstracción, encapsulamiento y modularidad son sinérgicos: un objeto proporciona una frontera bien definida alrededor de una sola abstracción, y tanto el encapsulamiento como la modularidad proporcionan barreras que rodean a esa abstracción. En cualquier caso, encontrar las clases y los objetos correctos y organizarlos después en módulos separados son decisiones de diseño independientes. La identificación de clases y objetos es parte del diseño lógico de un sistema, mientras que la identificación de los módulos es parte del diseño físico del mismo. No pueden tomarse todas las decisiones de diseño lógico antes de tomar todas las de diseño físico y viceversa, sino que estas decisiones de diseño se dan de forma iterativa.

Jerarquía

Una jerarquía es una clasificación de las abstracciones. Como hemos visto anteriormente, las dos jerarquías más importantes en un sistema complejo son la jerarquía de generalización/especialización, que define relaciones ES_UN y la jerarquía de agregación, que define relaciones ES_PARTE_DE.

Las jerarquías de generalización/especialización también se conocen como *herencia*. Básicamente, la herencia define una relación entre clases, en las que una clase comparte la estructura de comportamiento definida en una o más clases (herencia simple

y herencia múltiple, respectivamente). La herencia representa así una jerarquía de abstracciones, en las que una subclase hereda de una o más superclases. Típicamente, una subclase aumenta o redefine la estructura y el comportamiento de sus superclases.

A medida que se desarrolla la jerarquía de herencias, aquellas estructuras y comportamientos comunes a diferentes clases tenderán a migrar hacia superclases comunes. Las superclases representan abstracciones generalizadas y las subclases representan especializaciones en las que los campos y métodos de la superclase experimentan añadidos, modificaciones, o incluso ocultaciones. Así, la herencia permite declarar las abstracciones con economía de expresión.

Podemos suponer el siguiente ejemplo: en una universidad hay profesores y alumnos, ambos comparten características comunes (como los nombres, apellidos, dirección, edad, etc.). Estas características comunes pueden situarse en una clase genérica, denominada Persona. De esta clase genérica heredarán dos clases más específicas, Estudiante y Profesor, que contendrán respectivamente las características propias de los estudiantes y de los profesores (Figura 1.14).

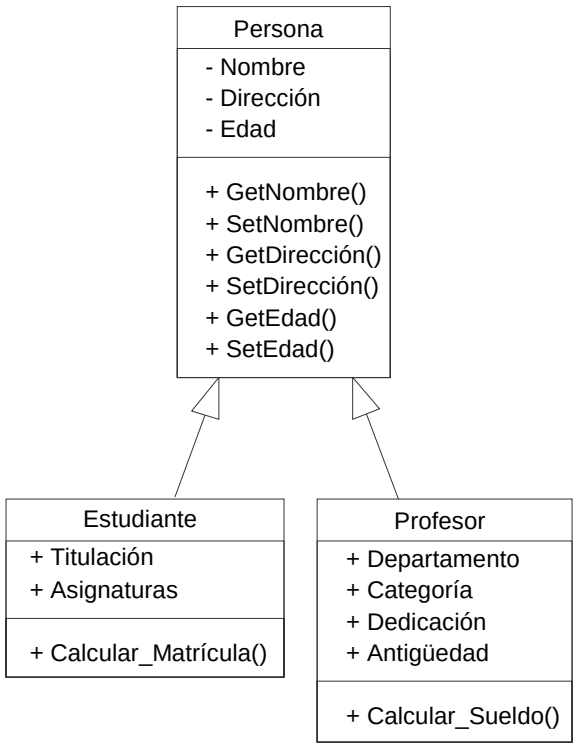


Figura 1.14 Ejemplo de herencia

En este ejemplo también podemos observar cómo funciona la abstracción y el encapsulamiento. Así, sólo se definen las características de las entidades que son relevantes para el problema en cuestión, dentro de las clases se definen atributos y métodos relacionados con dichos atributos y se oculta información al exterior. La clase persona se ha definido siguiendo la ortodoxia de la orientación a objetos, los atributos se declaran privados de forma que sólo sean accesibles al exterior a través de métodos de acceso de lectura (Get) y escritura (Set). Este procedimiento permite controlar el acceso

a los atributos aunque por motivos de simplificación en ocasiones se permite el acceso directo a los atributos (como en las clases Estudiante y Profesor).

Si realizamos una pequeña reflexión sobre los conceptos que hemos visto hasta el momento, vemos que existe una cierta tensión entre encapsulamiento y jerarquía. El encapsulamiento intenta proporcionar una barrera opaca tras la que ocultar los métodos y el estado, mientras que la herencia requiere abrir esta interfaz en cierto grado y puede permitir el acceso a los métodos y al estado a sus subclases pero no a otro tipo de clases.

Esto responde a un principio fundamental de la técnica de descomposición modular y que se conoce como **principio abierto-cerrado**. Este principio simplemente dice que los módulos deben ser a la vez abiertos y cerrados. La contradicción entre estos dos términos es sólo aparente en tanto que corresponden a dos objetivos de naturaleza diferente:

Un módulo se llama abierto si está disponible para ser extendido. Por ejemplo, debería ser posible expandir su conjunto de operaciones o añadir campos a sus estructuras de datos. Esto es sencillo de hacer a través de la herencia.

Un módulo se llama cerrado si su descripción o interfaz es estable y está bien definido.

De esta forma se conjugan dos objetivos incompatibles: mantener los módulos abiertos a modificaciones posteriores y dar al sistema la estabilidad suficiente de forma que un cambio en un módulo no implique una reacción en cadena de cambios en muchos otros módulos que se basan, directa o indirectamente, en el módulo original.

Para facilitar la aplicación del principio abierto-cerrado los atributos y métodos de una clase se pueden declarar “protegidos”, esto significa que son invisibles para las clases externas pero visibles para las subclases, lo que facilita la extensión de la clase original a través de la herencia.

Polimorfismo

Polimorfismo significa literalmente “la capacidad para adoptar varias formas”. En el desarrollo orientado a objetos el polimorfismo puede aparecer de varias formas, la más común de ellas es aquella en la que a una variable de una superclase se le asigna un objeto perteneciente a una de sus subclases. De esta forma, y siguiendo el ejemplo anterior, podríamos tener un “array” de variables de tipo Persona (denominado Listado) a las que se le asignan indistintamente objetos Estudiante o Profesor. A continuación representamos el código Java que representa esta situación:

```
Persona Listado[]= new Persona[10]; //Definimos un array de 10 Personas

Listado [0] = new Estudiante (); // Inicializamos los elementos del Listado
Listado [1] = new Profesor ();
Listado [2] = new Profesor ();
Listado [3] = new Estudiante ();
```

La potencia de esta aproximación estriba en que se pueden añadir nuevos tipos de personas (por ejemplo, personal de administración y servicios, becarios) sin tener que retocar el código existente, con lo cual la aplicación es fácilmente ampliable.

El polimorfismo se suele combinar con otra característica, denominada ligadura dinámica, que permite decidir qué método se aplicará a un determinado objeto en tiempo de ejecución y no en tiempo de compilación. Así, el sueldo de una persona se calcula de forma diferente según que persona se trate, para los becarios depende de su tipo; para los profesores de su categoría, antigüedad y dedicación; para los PAS depende de su categoría. Para aquellas subclases que no tengan sueldo se define un método Calcular_Sueldo en la clase Persona que devuelve cero, este método se redefinará en aquellas clases que convenga (Figura 1.15).

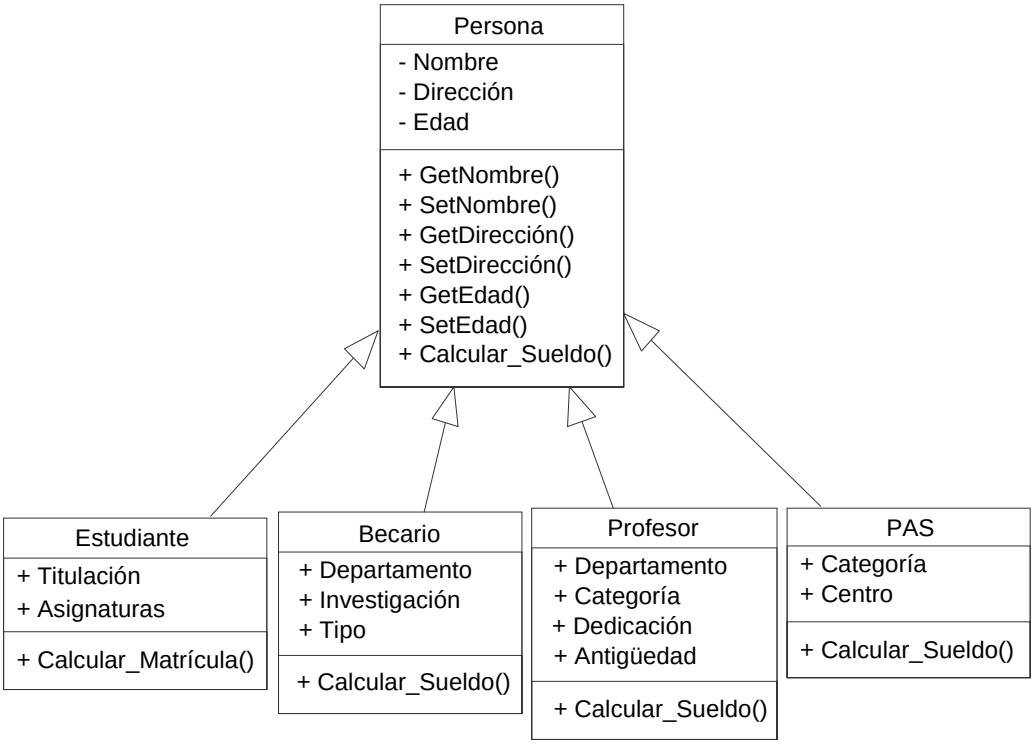


Figura 1.15 Ejemplo en el que el método genérico Calcular_Sueldo se redefine por aquellas subclases que lo necesiten

Ahora podemos aplicar el método Calcular_Sueldo sobre el objeto persona, gracias al polimorfismo y a la ligadura dinámica si el objeto Persona contiene un profesor se ejecutará el método Calcular_Sueldo de la clase profesor. El código Java correspondiente sería:

```
Persona Listado[]= new Persona[10]; //Definimos un array de 10 Personas

Listado [0] = new Estudiante (); // Inicializamos los elementos del Listado
Listado [1] = new Profesor();
Listado [2] = new PAS();

Listado[0].Calcular_Sueldo();    // Llama a Calcular_Sueldo de Persona
Listado[1].Calcular_Sueldo();    // Llama a Calcular_Sueldo de Profesor
Listado[2].Calcular_Sueldo();    // Llama a Calcular_Sueldo de PAS
```

Otra forma de polimorfismo es lo que se conoce como sobrecarga. Decimos que el nombre de un método está sobrecargado cuando a dicho nombre se le pueden asignar distintos cuerpos. La forma de determinar que cuerpo del método hay que ejecutar en cada momento puede hacerse por los parámetros del método (sobrecarga paramétrica) o por la clase a la que pertenece el método (sobrecarga de mensajes).

Un ejemplo típico de sobrecarga paramétrica es la operación de la suma, en la que si sus parámetros son números enteros realiza una suma aritmética pero si sus parámetros son cadenas de caracteres realiza la concatenación de las mismas. Un ejemplo típico de sobrecarga de mensajes sería el método "dibujar" que puede utilizarse para dibujar una ventana, un botón, una lista desplegable, etc. según desde qué clase se llame al método.

Ventajas e inconvenientes de la orientación a objetos

El paradigma de la orientación a objetos presenta varias ventajas entre las que podemos citar las siguientes:

Flexibilidad y extensibilidad. La combinación de las características de herencia, polimorfismo, ligadura dinámica, etc. hacen posible utilizar y definir de forma clara módulos funcionalmente incompletos que permiten su extensión sin trastornar la operación de otros módulos o de sus clientes. De esta forma los sistemas son flexibles, fácilmente extensibles y de mantenimiento menos costoso.

Reutilización. Los objetos bien diseñados constituyen la base para otros sistemas que se ensamblan, en gran parte a partir de módulos reutilizables, lo que redunda en una mayor productividad.

Escalabilidad. La combinación de la reutilización con la extensibilidad permite desarrollar nuevos sistemas complejos a partir de partes ya desarrolladas, lo que reduce la complejidad del desarrollo.

Naturalidad. La aproximación orientada a objetos corresponde a una visión más natural del mundo real que los típicos diseños top-down que se basan en una descomposición funcional por refinamiento progresivo.

Seguridad. La ocultación de la información contribuye a la construcción de sistemas seguros.

Sin embargo, esta técnica de representación del conocimiento no está exenta de inconvenientes entre los que citamos:

Rendimiento. El empleo profuso de la herencia, el polimorfismo y la ligadura dinámica ofrece mucha potencia al desarrollo pero por su complejidad pueden afectar al rendimiento global del sistema al consumir más recursos (memoria, tiempo, etc.)

Problemas de reutilización. La reutilización siempre se ha considerado una de las ventajas principales de la orientación a objetos. Sin embargo, es fácil caer en diseños ad-hoc no pensados para su reutilización posterior. Un diseño reutilizable necesita una consideración especial y un coste de desarrollo mayor.

Cambio de cultura. La aproximación a objetos obliga a pensar en términos de objetos y mensajes, mientras que la programación habitual está pensada en términos de datos y funciones. El cambio de puntos de vista no siempre es fácil.

1.5. Reglas de Producción

Las reglas de producción (o simplemente reglas), son esquemas de representación del conocimiento que pertenecen a lo que hemos denominado métodos procedimentales de representación. Recordemos que en los métodos procedimentales la mayor parte de los conocimientos se representan como procedimientos dinámicos. Estructuralmente, las reglas de producción son elementos de representación del conocimiento dinámico constituidas por una parte IF (denominada condición o premisa), una parte THEN (denominada conclusión o acción) y, opcionalmente, una parte ELSE (o conclusión-acción alternativa que se ejecuta cuando la premisa es falsa).

IF <CONDICION>
THEN <CONCLUSION O ACCION>

IF <CONDICION>
THEN <CONCLUSION>
ELSE <ALTERNATIVA>

Básicamente, la premisa de una regla está constituida por un conjunto de cláusulas que pueden estar anidadas a través de los juntores u operadores de relación AND, OR, NOT:

IF: cláusula A, AND cláusula B, OR cláusula C, ...

Un primer problema que aparece cuando empleamos reglas de producción con cláusulas anidadas es conseguir que la premisa represente, exactamente y de forma precisa, la heurística correspondiente. Así, no es lo mismo:

IF: [(A OR B) AND (C OR NOT D)]

que

IF: [A OR (B AND C) OR NOT D]

Una vez definimos la estructura clausal de problema, el siguiente paso consiste en encontrar una representación interna adecuada, tanto para las cláusulas, como para las acciones y las alternativas. Aunque este proceso es dependiente de la herramienta que utilicemos, o del lenguaje que empleemos, está claro que dicha representación interna debe ser compatible con la representación elegida para el conocimiento estático o declarativo.

Normalmente, dicha representación se realiza a través de ternas <parámetro/relación/valor>, en donde los parámetros son las características que queremos investigar. Los parámetros deben compararse con los valores correspondientes a través del operador de relación definido, al objeto de averiguar si la cláusula es cierta o no.

Por otra parte, la parte THEN de la regla, que puede ser múltiple[1], suele representar una hipótesis de trabajo (que se constata cuando la premisa es cierta), o una acción que puede ser ejecutada. Veamos el siguiente ejemplo:

```
IF    :   (1)   La presión arterial sistólica es mayor de 160 mmHg,
AND   :   (2)   La presión arterial diastólica es mayor de 95 mmHg,
AND   :   (3)   La presión arterial media es mayor de 130 mmHg
THEN  :   (1)   El paciente presenta hipertensión arterial,
AND   :   (2)   Actualizar la base de datos con la conclusión
```

La regla anterior, descrita en lenguaje natural, podría traducirse del siguiente modo:

```
(HEMODYN-1)
IF    :   (1)   (Presión_arterial sistólica)  gt 160
AND   :   (2)   (Presión_arterial diastólica) gt 95
AND   :   (3)   (Presión_arterial media) gt 130
THEN  :   (1)   (Diagnóstico hemodinámico hipertensión_arterial)
AND   :   (2)   (ACTUALIZA
                '(Diagnóstico hemodinámico hipertensión_arterial)
                'Base_de_datos)
```

El ejemplo anterior muestra una regla con parámetros numéricos. Una regla con parámetros de naturaleza literal podría ser la siguiente:

```
(GASOM-1)
IF    :   (1)   (Gases_arteriales CO2)  eq   hypercapnia
AND   :   (2)   (Gases_arteriales pH)   eq   acidemia
AND   :   (3)   (Gases_arteriales Bic)  eq   Normal
THEN:     (1)   (Diagnóstico Respiratorio Acidosis_respiratoria)
```

Los dos ejemplos anteriores nos permiten ilustrar cómo las reglas pueden cooperar con otras estructuras de representación declarativa del conocimiento, por ejemplo frames. Efectivamente, supongamos que tenemos la siguiente frame que representa la situación actual de un determinado caso:

```
Presión_arterial
        Sistólica
                177
        Diastólica
                99
        Media
                121
```

[1] Y por lo tanto puede estar constituida por varias acciones.

Si sobre esta frame, y tras un proceso de emparejamiento que ya comentaremos, ejecutamos la regla HEMODYN-1, la conclusión podría representarse por medio de la frame:

Diagnóstico
Hemodinámico
Hipertensión_arterial

Nuevamente, si disponemos de la frame:

Gases_arteriales
CO2
Hypercapnia
pH
Acidemia
Bic
Normal

la aplicación de la regla GASOM-1 sobre la frame de datos anterior produce el siguiente resultado:

Diagnóstico
Respiratorio
Acidosis_respiratoria

Obviamente, la aplicación de ambas reglas sobre los datos disponibles produce el siguiente resultado:

Diagnóstico
Hemodinámico
Hipertensión_arterial
Respiratorio
Acidosis_respiratoria

En algunos casos de anidamientos homogéneos de cláusulas es útil definir y considerar distintos tipos de reglas: reglas IFALL, reglas IFSOME, y reglas IFANY.

En las reglas IFALL, todas las cláusulas de la premisa han de ser ciertas para que se ejecute la acción o se establezca la conclusión de la parte THEN. Una regla IFALL equivale a una regla en la que todas las cláusulas estén anidadas por medio de operadores AND:

IFALL:	A es cierto		IF:	A es cierto
	B es cierto		and:	B es cierto
	C es falso	equivale a ...	and:	C es falso
THEN:	H es rojo		THEN:	H es rojo

La estructura de los hechos de la regla anterior, para que la premisa sea cierta, debe ser la siguiente:

[< A=t.>, < B=t >, < C=f >], con “t” y “f” booleanos (i.e., t=true, f=false)

Las reglas tipo IFANY y/o IFSOME equivalen a reglas en las que las cláusulas de la premisa estén conectadas por medio de operadores OR. Algunos autores encuentran diferencias entre ambos tipos de reglas. Así, para que la premisa de una regla IFANY sea cierta basta con que una cláusula sea cierta.

De este modo, en cuanto se encuentra una cláusula que verifica la condición ya no es preciso investigar más y la acción se ejecuta o la hipótesis se establece. Por el contrario, en una regla de tipo IFSOME se investigan todas las cláusulas antes de ejecutar la acción o establecer la hipótesis (que se establece, igual que en una regla IFANY, cuando al menos una cláusula es cierta). Al respecto, una regla IFANY formaliza una búsqueda no exhaustiva mientras que una regla IFSOME representa una búsqueda exhaustiva. Este planteamiento diferencial entre ambos tipos de reglas puede tener importantes repercusiones en razonamientos imprecisos, como más adelante veremos.

Si consideramos diferencias entre reglas IFANY y reglas IFSOME, la estructura de los hechos representados también cambia. Así, mientras una regla de tipo IFANY considera un OR exclusivo, una regla de tipo IFSOME implica un OR inclusivo, como se muestra a continuación:

IFANY:	(1)	A es cierto
	(2)	B es falso
THEN:	(1)	V es bueno $\rightarrow$ [< A=t >] or [< B=f >]

IFSOME:	(1)	A es cierto
	(2)	B es falso
THEN:	(1)	V es bueno $\rightarrow$ [< A=t >] or [< B=f >] or [< A=t > and < B=f >]

Nótese que el hecho de definir distintos tipos de reglas (i.e., IFALL, IFANY, IFSOME), no excluye en absoluto la posibilidad de nuevos anidamientos, y estructuras como la que se muestra a continuación serían perfectamente válidas:

IFALL:	1.- OR	
	1.1.-	A es verdadero
	1.2.-	NOT C = desconocido
	2.- NOT	V = blanco
	3.- Luis = alto	
THEN:	1.- H = Hipótesis	

La representación de conocimiento mediante reglas de producción presenta ciertas ventajas. Así, las condiciones y las acciones involucradas son explícitas. Además, el conocimiento es representado de forma muy modular ya que cada regla de un sistema dado constituye una unidad completa de conocimiento. Por último, las reglas de producción permiten almacenar y utilizar conocimiento de una gran especificidad y el conocimiento implicado suele ser de naturaleza heurística.

1.6. Resumen

En este capítulo hemos descrito algunos de los métodos estructurados de representación del conocimiento, que introducimos como una alternativa a los procedimientos de representación más formales. Tras una breve mención a las características esenciales que debemos exigir a cualquier método estructurado de representación del conocimiento, pasamos a su clasificación en "métodos declarativos" y "métodos procedimentales". En los métodos declarativos se da más importancia a los hechos y entidades del dominio que a los mecanismos de manipulación de los mismos. Por el contrario, los métodos procedimentales, aunque operen sobre hechos y entidades del dominio de discurso, prestan mayor atención a los mecanismos de relación entre entidades. Los métodos declarativos estudiados han sido las redes semánticas -en las que el conocimiento se representa como una colección de nodos unidos entre sí por medio de arcos etiquetados-, los modelos de dependencia conceptual -que utilizan conjuntos de primitivas conceptuales y las combinan para obtener significados en cualquier lenguaje-, las frames -que pueden definirse como redes semánticas complejas que tratan el problema de la representación desde la óptica del razonamiento por semejanza-, y los guiones -que son especializaciones del concepto de frame capaces de representar adecuadamente secuencias típicas de sucesos-. También, y en relación a los métodos declarativos de representación, hemos comentado el importante concepto de herencia de propiedades, por medio de la cual un determinado objeto adquiere las propiedades de la clase a la que pertenece. Finalizamos el tratamiento de los métodos declarativos con una breve descripción del paradigma de orientación a objetos. Por otra parte, como método procedimental de representación del conocimiento, hemos descrito el formalismo de las reglas de producción, poniendo especial atención en su estructura y su forma de "cooperar" con algunas declarativas de representación (e.g., frames).

1.7. Textos Básicos

Maté, Pazos, "Ingeniería del Conocimiento: Diseño y Construcción de Sistemas Expertos", CETTICO, eds., 1988

Rich, "Inteligencia Artificial", G.Gili, eds., 1988

Rich, Knight, "Inteligencia Artificial", McGraw-Hill, eds., 1994

Rolston, "Principios de Inteligencia Artificial y Sistemas Expertos", McGraw-Hill, eds., 1990

González, Dankel, "Verification and Validation", En: The engineering of knowledge-based systems : theory and practice, Prentice-Hall International, 1993.