TESIS DOCTORAL

Metamodelización y Formalismos para la Representación del Comportamiento

Ángel Luis Rubio García

UNIVERSIDAD DE LA RIOJA

TESIS DOCTORAL

Metamodelización y Formalismos para la Representación del Comportamiento

Ángel Luis Rubio García

Universidad de La Rioja
Servicio de Publicaciones
2003

Esta tesis doctoral, dirigida por el doctor D. Eladio Domínguez Murillo fue leida el 18 de Junio de 2002, y obtuvo la calificación de Sobresaliente cum Laude por unanimidad

Edita: Universidad de La Rioja
Servicio de Publicaciones

ISBN 84-688-0890-3

Universidad de La Rioja
Departamento de Matemáticas y Computación

Metamodelización y Formalismos para la Representación del Comportamiento

Tesis Doctoral

Doctorando: Ángel Luis Rubio García
Director: Dr. D. Eladio Domínguez Murillo
Logroño, Marzo 2002

La esencia de mi monstruosidad no puede ser percibida sensorialmente por los demás, y ni yo mismo soy capaz de ello. Pero, evidentemente, ha sido establecida de un modo empírico, y por lo tanto, desde un punto de vista fenomenológico, mi calidad de monstruo no es un asunto de simple apriorismo, aunque en el plano existencial haya tenido como resultado la revelación de mi verdadera situación en el mundo.

Philip Kerr *Una investigación filosófica*

La realización del presente trabajo ha sido financiada parcialmente por las ayudas ATUR97/49, ATUR98/26, y ATUR99/31 y los Proyectos de Investigación DGES PB96–0098–C04–01 y DGES PB98–1621–C02–01.

Agradecimientos

En la actitud ante el investigar,
un mero trabajo se puede dar
o un amar lo que se quiere crear.
Con un mero trabajar,
sólo eso se hará,
trabajar.
Pero con el otro actuar,
el amor como vínculo se sentirá,
como vínculo a lo que creado será,
como fuerza que conducirá
a lo que inventado será.

Eladio Domínguez – *A Ángel Luis* (Marzo, 1999)

Decir que la elaboración de una tesis doctoral es un proceso largo, intermitente en intensidad y difícil en muchos momentos no es más que constatar algo obvio para cualquiera que haya alcanzado el grado de Doctor. Decir que el esfuerzo que supone culminar esta tarea resulta esencial para madurar, tanto en el ámbito profesional como personal, es seguramente una reflexión que otros muchos habrán hecho en circunstancias similares. Pero el tópico que con seguridad es el más repetido, y sin lugar a dudas el más cierto cuando se refiere a trabajos que se extienden en el tiempo, es que uno jamás podría haber llegado al resultado final sin la complicidad, la paciencia y la ayuda de un amplio grupo de personas. El apoyo de todos y cada uno, expresado en una multitud de formas diferentes, ha sido fundamental para llegar hasta aquí y es no sólo una obligación, sino un auténtico placer poder expresar mi más sincero agradecimiento.

En primer lugar, gracias a Eladio, que cumpliendo fielmente su auto-adjudicado papel de"no-jefe" me ha dejado tropezar (¡qué gran manera de aprender!) en las ocasiones necesarias, pero que ha guiado la nave con gran visión y previsión, cambiando el rumbo en los momentos oportunos. Aunque la distancia geográfica haya supuesto algunos contratiempos para el avance de la investigación en momentos puntuales, bien está lo que bien acaba, y más cuando lo que hemos puesto es tan

sólo un punto y seguido. Los versos fenomáticos dedicados por Eladio con los que he comenzado estas páginas reflejan sin duda el espíritu del trabajo que hay detrás de lo escrito aquí y auguran el espíritu del trabajo que vendrá. ¡Ah, y gracias por haber descubierto a tiempo que los primeros pasos en la gestación de la Fenómatica eran demasiado para mí!

Gracias también al resto de integrantes del Grupo de Investigación Nóesis de la Universidad de Zaragoza, pues su estilo y forma de trabajar han influido decisivamente en el mío. En particular, gracias a Inés, ya que parte del presente trabajo está basado en el suyo, y sobre todo, gracias a Toñi, puesto que bajo su co-dirección y orientación hemos culminado la etapa de investigación más fructífera y brillante de mi corta carrera.

Quiero agradecer a la Universidad de La Rioja su apoyo en medios materiales. Y dentro de ella tengo que destacar al Departamento de Matemáticas y Computación (al que tengo el orgullo de pertenecer) y no sólo por los recursos facilitados y la confianza mostrada en todos estos años, sino sobre todo por los medios humanos con los que cuenta. Resulta difícil encontrar e incluso imaginar una atmósfera de trabajo mejor que la que se disfruta en este Departamento, y sin duda los responsables de esta situación son todos y cada uno de sus miembros. Ellos hacen que la convivencia sea tan grata, que el trabajo diario sea mucho más llevadero, y que como consecuencia este Departamento sea envidiado dentro y fuera de la comunidad universitaria. Aunque reconozco que es algo injusto resaltar a unos por encima de otros, sí quiero agradecer a Laureano su apoyo desde el primer día (el día en que orientó mi vocación, guiándome hasta el Seminario... para firmar mi primer contrato); quiero rememorar los muchos años de despacho y trabajo compartidos con Francisco, con quién por suerte o por desgracia sigo compartiendo esta puñetera obsesión perfeccionista; y quiero recordar a Vico, con quien por haber sido paralelo nuestro "tiempo de tesis", he podido compartir los momentos de crisis –y también los de euforia– que inevitablemente suceden.

Este rápido trayecto desde la Universidad de Zaragoza hacia la Universidad de La Rioja me conduce inevitablemente hasta una de las personas que, como no podía ser de otro modo, más han tenido que ver en el resultado final (y no estoy hablando sólo de investigación) de todo el trabajo realizado. Desde el inicial e intencionado distanciamiento investigador, hasta la profunda implicación final (empujón definitivo incluido), pasando por el continuo seguimiento, colaboración e interés por mí a lo

largo de los años, Julio ha supuesto una constante ayuda, un apoyo al que recurrir directa o indirectamente. Y aunque quizá de modo inconsciente he intentado no reflejarme demasiado en tan brillante espejo, un modelo (¿un metamodelo?) tan cercano influye, y siempre y por suerte, influye para mejorar.

Mi hermano supone el obvio y agradable enlace entre mi mundo laboral, investigador y mi mundo familiar. Gracias a toda mi familia por su constante preocupación, y muy especialmente a mi madre, siempre inquieta porque su chiquillo pequeño recibiera como mínimo el mismo trato que los demás. Gracias también por ello, y gracias por los esfuerzos para viajar a Logroño y así facilitar mis desplazamientos a Zaragoza. Los mismos o parecidos esfuerzos los han hecho también mis suegros (¿Dónde va Perico?: donde va Juanico). En el fondo todos sabemos que para los tres sus viajes han sido, son y serán más un placer que un inconveniente.

Y termino con los que he dejado para el final pero que son también el comienzo, el punto medio y el todo. Mis verdaderas razones para empezar, mis verdaderas razones para seguir y mis verdaderas razones para por fin terminar, sois vosotros. El final de este trabajo ha coincidido con el principio de vuestras vidas, Adrián, Víctor, y en estos años he podido disfrutar de y con vosotros, aunque a veces haya tenido la sensación de no habeos dedicado todo el tiempo que hubiera querido. Por suerte, ese posible hueco lo has llenado tú, Loli, con tu infinito apoyo, tu comprensión, tu paciencia, tu amor en definitiva. Estoy convencido de que al final todos los sacrificios han merecido la pena: este trabajo está dedicado, nunca mejor dicho, a vosotros tres.

Introducción

Actualmente existen multitud de metodologías, técnicas, lenguajes, métodos, herramientas, etc. que pueden ser utilizadas durante el desarrollo de sistemas software complejos. En particular, un número importante de estas técnicas y lenguajes están involucrados en la representación de aspectos relativos al comportamiento de dichos sistemas, y la inmensa mayoría de las metodologías de análisis y diseño orientados a objetos incluyen componentes para la modelización estructural junto con componentes para la modelización de comportamiento. Esta situación es particularmente relevante cuando la naturaleza del sistema que se ha de modelizar es esencialmente dinámica, tal y como ocurre por ejemplo en los sistemas reactivos o en los sistemas de tiempo real. Para la modelización de este tipo de sistemas se han desarrollado distintos formalismos específicos, tales como Statecharts o Redes de Petri, para los que a su vez se han elaborado multitud de variantes. Esta compleja situación sugiere la conveniencia de disponer de un marco que permita describir los conceptos esenciales ligados a la representación de comportamiento con independencia del lenguaje o técnica concreta que se utilice. Este marco permitiría el estudio detallado de estos lenguajes o técnicas, siendo este un paso previo para el análisis posterior de problemas tales como comparación, adaptación, transformación, etc. de dichos lenguajes. En este trabajo se presenta una solución para este problema, a través de la introducción de una arquitectura genérica, denominada arquitectura NÓesis. Como medio para abstraer las particularidades de cada lenguaje o técnica concreta, para el desarrollo de la arquitectura utilizaremos una perspectiva de metamodelización. Se ha demostrado que el uso de la perspectiva de metamodelización es un medio válido para mejorar la usabilidad, comprensibilidad y legibilidad durante el estudio (análisis, diseño, comparación, adaptación, etc) de lenguajes y técnicas. En concreto en este trabajo utilizamos una técnica de metamodelización concreta, la técnica NÓesis, ilustrándola en primer término mediante un ejemplo

de metamodelo del modelo de bases de datos RM/T (Capítulo 2). El uso de esta técnica junto con las directrices que proporciona la arquitectura Nóesis (Capítulo 3) nos han permitido desarrollar un metamodelo de Statecharts que recoge toda la potencia expresiva de este formalismo, tanto en un sentido más cercano a los aspectos sintácticos como en los aspectos puramente de comportamiento (Capítulo 4). Como prueba de la versatilidad de la arquitectura Nóesis, en este trabajo se incluye también un metamodelo de UML State Machines, la versión orientada a objeto de Statecharts que ha sido incluida dentro de UML. La particularidad de este otro metamodelo es que para su expresión se ha utilizado como lenguaje de metamodelización el propio UML, siguiendo el estilo "oficial" de definición de este lenguaje (Capítulo 5). Estos ejemplos prueban que la utilización de la arquitectura Nóesis es independiente de la perspectiva de metamodelización utilizada, siendo por tanto una herramienta flexible para la representación de aspectos de comportamiento. La presentación detallada de la estructura de la memoria está incluida en el Capítulo 1, el cual presenta también el contexto general en el que se enmarcan nuestras aportaciones.

Índice general

Agradecimientos v

Introducción ix

Índice general xi

Índice de figuras xiii

Índice de tablas xv

1. Preliminares 1

1.1. ¿Modelar o modelizar? 2
1.2. Una(s) idea(s) de modelo 3
1.3. Modelos, lenguajes y comunicación 9
1.4. Una(s) idea(s) de lenguaje 12
1.4.1. Algunos tipos de lenguajes en Informática 15
1.5. Modelización de modelos y de lenguajes: metamodelización 18
1.5.1. Niveles de modelización y metamodelización 22
1.5.2. Metamodelización avanzada: Ingeniería del Método 29
1.6. Perspectiva general de la memoria 35

2. Un metamodelo Nóesis de RM/T 41

2.1. Introducción a la técnica de metamodelización Nóesis 42
2.1.1. Características principales de la técnica Nóesis 46
2.1.2. Otras características de la técnica Nóesis 67
2.2. Un metamodelo Nóesis de RM/T 68
2.2.1. Referencias sobre RM/T 68
2.2.2. Un metamodelo Nóesis de RM/T: parte intensional básica . . 72

2.2.3. Un metamodelo NÓESIS de RM/T: parte extensional básica . 96
2.2.4. Aspectos avanzados de RM/T 122

3. Metamodelización de formalismos de comportamiento 137
3.1. Formalismos de comportamiento: estática y dinámica 138
3.2. La arquitectura NÓESIS para la representación del comportamiento . 146
3.2.1. La arquitectura en el Nivel de Aplicación 148
3.2.2. La arquitectura en el Nivel del Método 153
3.2.3. La arquitectura en el Nivel Axiomático 159
3.2.4. Refinamiento de la arquitectura NÓESIS 161
3.2.5. Ventajas de la utilización de la arquitectura NÓESIS 164
3.3. La noción de metamodelo de comportamiento NÓESIS 166

4. Un metamodelo Nóesis de Statecharts 171
4.1. Introducción al metamodelo 172
4.2. Perspectiva y Sistema de Referencia 174
4.3. Conceptos básicos del metamodelo: estado y transición 180
4.4. Marco representacional 188
4.5. Definición de modelo 189

5. Un metamodelo UML de UML State Machines 197
5.1. Introducción al metamodelo 198
5.2. Un metamodelo UML de UML State Machines 201
5.2.1. Nivel de Independencia de Status 201
5.2.2. Nivel de Status 205
5.2.3. Transformaciones T_0 y T 208
5.3. Trabajos relacionados y cuestiones abiertas 211

Conclusiones 217

Bibliografía 221

Índice de figuras

1.1. Arquitectura de cuatro niveles 19
1.2. Arquitectura de metamodelización en zigzag 24
1.3. Arquitectura de metamodelización anidada 25
1.4. Dimensiones de modelización 26
1.5. Interpretación de dimensiones de modelización 28

2.1. Dimensiones de modelización tratadas en la técnica NÓESIS 43
2.2. Ejemplo de soporte. Soporte de fórmulas 64
2.3. Nociones de E-dominio y [E]-dominio 78
2.4. Soporte parcial. Atributos en RM/T 80
2.5. Soporte parcial. Esquemas de relación en RM/T 84
2.6. Soporte parcial. Esquemas de relación en RM/T. Notación alternativa 85
2.7. Soporte parcial. Esquemas de molécula en RM/T 92
2.8. Soporte parcial. Parte intensional de bases de datos RM/T 95
2.9. Soporte parcial. Valores de atributos en RM/T 101
2.10. Soporte parcial. Tuplas en RM/T 102
2.11. Soporte parcial. Relaciones en RM/T 104
2.12. Soporte parcial. Moléculas en RM/T 111
2.13. Soporte. Base de datos RM/T 114
2.14. Diagrama E/R para el ejemplo de personas propietarias de coches . . 127
2.15. Diagrama E/R para el ejemplo de personas tomadores de seguro de coches 127

3.1. Statechart para un Sistema X: Vista Estática 139
3.2. Statechart para el Sistema X con estado A activo 140
3.3. Statechart para el Sistema X con transición t1 disparable 141

3.4. Statechart para el Sistema X con estado B activo 142
3.5. Red de Petri para el Sistema X: Vista Estática 144
3.6. Red de Petri para el Sistema X con lugar A marcado 145
3.7. Red de Petri para el Sistema X con lugares A y e marcados 146
3.8. Representación gráfica de la Arquitectura NÓESIS 147
3.9. Statechart para el termostato 149
3.10. Statechart para el termostato en situación inicial 151
3.11. Refinamiento de la transformación T 164

4.1. Soporte Independiente de Status del Metamodelo NÓESIS de Statecharts 187
4.2. Soporte de Status del Metamodelo NÓESIS de Statecharts 191
4.3. Soporte de Status Virtual 1 del Metamodelo NÓESIS de Statecharts . 195
4.4. Soporte de Status Virtual 2 del Metamodelo NÓESIS de Statecharts . 195
4.5. Soporte de Status Virtual 3 del Metamodelo NÓESIS de Statecharts . 196

5.1. Diagrama Estático del metamodelo de UML State Machines 202
5.2. Diagrama Dinámico del metamodelo de UML State Machines 207
5.3. Primer Diagrama Intermedio del metamodelo de UML State Machines 212
5.4. Segundo Diagrama Intermedio del metamodelo de UML State Machines 213

Índice de tablas

2.1. Notación NÓESIS para la representación textual de conceptos 49
2.2. Notación NÓESIS para la representación gráfica de soportes 57
2.3. Perspectiva del metamodelo NÓESIS de RM/T 73
2.4. Sistema de referencia del metamodelo NÓESIS de RM/T 115
2.5. Restricciones locales del metamodelo NÓESIS de RM/T 119
2.6. Restricciones globales del metamodelo NÓESIS de RM/T 121

4.1. Perspectiva del metamodelo NÓESIS de Statecharts 175
4.2. Sistema de referencia del metamodelo NÓESIS de Statecharts 175
4.3. Restricciones locales del metamodelo NÓESIS de Statecharts 188
4.4. Transformaciones del metamodelo NÓESIS de Statecharts 193

5.1. Transformaciones del metamodelo de UML State Machines 209

Capítulo 1

Preliminares

En este apartado introductorio vamos a tratar de centrar el contexto en el que se encuadra el resto del presente trabajo. Ciertamente resulta difícil encontrar un área de conocimiento en la que englobar los resultados que iremos presentando a lo largo de los próximos capítulos, pero, en una primera aproximación, podemos afirmar que esta memoria se enmarca en general dentro de la Informática, de las Ciencias de la Computación, más en particular en el ámbito del desarrollo de *sistemas de información*, y en concreto en los aspectos relacionados con la *modelización (o modelado)* de este tipo de sistemas. Un lector inquieto, ante este primer intento de contextualización, puede preguntarse, casi de manera automática: ¿Qué es un "sistema de información"? ¿Qué es "modelizar"(o modelar)? Contestar, de manera precisa, a estas preguntas es una tarea ardua, por no decir imposible. Por ejemplo, el término "sistema de información" puede tener diferentes connotaciones, en función del tipo de persona que trate de interpretar dicho término. Así, se puede ver un sistema de información como un sistema *técnico* (relacionado con la informática y las ciencias de la computación, las telecomunicaciones, la gestión de datos, etc.), como un sistema *social* (relacionado con las necesidades de información de algún tipo de organización humana), como un sistema *conceptual* (proveniente de la abstracción mental de alguno de los sistemas anteriores), etc. Existen multitud de referencias en la literatura que tratan sobre "sistemas de información" desde alguno (o algunos) de los puntos de vista anteriores, y es por tanto muy complicado tratar de obtener una descripción que pueda ser universalmente aceptada. De hecho, esta memoria no intenta tratar sobre sistemas de información en general, sino como hemos observado, sobre ciertos aspectos particulares relativos a la modelización (o modelado) de este

tipo de sistemas. A la vista de estas dificultades de partida, comenzaremos tratando de resolver la cuestión puramente terminológica que de manera implícita hemos planteado en algunas de las frases precedentes, cuestión que es sin duda un problema bastante más fácil de atacar.

1.1. ¿Modelar o modelizar?

¿Cuál es el término correcto, *modelar* o *modelizar*? Y consecuentemente, ¿debemos utilizar el término *modelado* o el término *modelización*? Aunque a estas alturas esta cuestión en concreto pueda parecer poco relevante, resulta sorprendente el que preguntas de este tipo se encuentren profundamente relacionadas con muchas partes del presente trabajo, como trataremos de ir mostrando en los próximos párrafos. En particular, la disyuntiva entre 'modelar' y 'modelizar' nos invita a consultar el diccionario; pues bien, ésta es una herramienta que nos hemos visto obligados a utilizar con frecuencia en esta memoria, y más aún, hemos tenido que revisar el concepto de lo que es un diccionario, adaptándolo a nuestras necesidades particulares. Volviendo al caso particular que nos ocupa, en el Diccionario de la Real Academia de la Lengua Española no existen entradas para los términos modelizar ni modelización, y sin embargo, en el Diccionario Collins Inglés-Español-Inglés encontramos ambas palabras, en el caso de modelizar prácticamente con idéntico sentido a modelar (to model) y en el caso de modelización casi como un sinónimo de modelado (modelling en inglés británico; modeling en el inglés de Estados Unidos). Esta aparente contradicción ocurre en muchas ocasiones cuando se trata de términos científicos o restringidos a una especialidad muy concreta. Puesto que muchos de estos términos proceden en su mayor parte del inglés, no se encuentran, en ocasiones, traducciones al castellano que sean adecuadas o que se acepten de forma general, por lo que difícilmente tales palabras pasan a formar parte del español "oficial". En el caso del vocablo anglosajón 'modelling', es cierto que su traducción más correcta, desde el punto de vista lingüístico, es 'modelado'. Sin embargo, aunque podemos encontrar traducciones recientes de libros escritos en inglés que utilizan el término modelado (como por ejemplo "Modelado y diseño orientados a objetos", traducción de [RBP+91] o "El lenguaje unificado de modelado. Manual de referencia", traducción de [RJB99]), también encontramos otros que utilizan el término modelización (como por ejemplo "Dominio de la modelización conceptual" [Pla92], si bien hay que señalar que en este

caso el libro original está escrito en francés, con título "Maîtriser. La Modélisation Conceptuelle"). Como última consideración acerca de la elección entre modelización y modelado, observemos que esta última palabra tiene en castellano una cierta connotación de representación física de algo (el modelado del barro, por ejemplo), mientras que la modelización está más relacionada con la manipulación de elementos esencialmente conceptuales. Además el modelado puede ir ligado a la consideración de un cierto valor estético o artístico. De alguna manera esta connotación también podría ser aplicable a la acción de creación de modelos dentro del ámbito de la Informática, de la misma forma que en algunos casos se habla de la belleza de algunos resultados en Matemáticas, pero no es habitual el considerar este aspecto estético en el ámbito de la Computación. A modo de resumen, diremos que, puesto que esta memoria va a estar centrada en las tareas relacionadas con la creación de modelos en Informática, y este tipo de modelos son en esencia modelos conceptuales, en este trabajo usaremos preferentemente los términos modelización y modelizar, si bien admitimos que es probable que lo más correcto –desde un punto de vista puramente lingüístico – fuera utilizar los términos modelado y modelar.

1.2. Una(s) idea(s) de modelo

Una vez abordado este pequeño divertimento terminológico, este ingenuo juego de palabras, podemos empezar a plantearnos preguntas algo más profundas. Asumiendo las acepciones más básicas, según las cuales 'modelización' es 'acción de modelizar', y a su vez 'modelizar' es 'crear un modelo', la siguiente cuestión es: ¿Qué es un modelo? Responder a esta pregunta de forma categórica no es desde luego un objetivo de esta memoria, ya que las implicaciones e interrelaciones de esta palabra con distintas áreas de las Ciencias de la Computación son inmensas, y sería necesario dedicar una memoria completa, y de manera exclusiva, a esta compleja cuestión. Puesto que hay que tomar algún punto de partida, vamos a consultar la abundante literatura sobre el tema para mostrar algo de luz sobre la idea de modelo. Tampoco en este caso nuestra intención es la de presentar una lista exhaustiva de las definiciones de modelo que los distintos autores han presentado, ya que es prácticamente imposible encontrar dos definiciones no ya coincidentes, sino ni siquiera equivalentes, al menos en su formulación, si bien es cierto que en la mayoría de los casos la intención de uso de las distintas definiciones sí que es la misma. Mostraremos cuatro definiciones de modelo,

exclusivamente a modo de ejemplo, sin intentar que supongan ni una clasificación ni una categorización de las distintas definiciones existentes en la literatura.

La definición de R. Planche en [Pla92]:

> Representación de un sistema que busca facilitar su comprensión, poniendo en evidencia ciertos aspectos o ciertos puntos e ignorando otros.

La definición de J. Rumbaugh y otros en [RBP+91]:

> Abstracción de algo, cuyo objetivo es comprenderlo antes de construirlo. Dado que los modelos omiten los detalles no esenciales es más sencillo manipularlos que manipular la entidad original.

La definición de D. Flynn y O. Fragoso en [FD96]:

> [De forma general,] una representación abstracta de una parte de la realidad. Usaremos el término *modelo* para referir a una representación abstracta de una parte de una organización que constituye un producto de la parte de modelización de información de un método.

Y por último la definición de J. Rumbaugh, I. Jacobson y G. Booch en [RJB99]:

> Representación en un cierto medio de algo del mismo u otro medio. El modelo captura los aspectos de la cosa que se modeliza desde un cierto punto de vista, y simplifica u omite el resto.

Siempre con la intención de responder de manera más precisa a la pregunta *¿qué es un modelo?*, parece interesante extraer los puntos claves que son compartidos por la mayoría de estas definiciones de ejemplo. Siguiendo en cierto modo el estilo periodístico, para responder el "qué" de un modelo, vamos a intentar resolver, al menos parcialmente, el "de qué y sobre qué" trata un modelo, "por y para qué"se crea un modelo y "cómo"se crea un modelo. Así, hemos agrupado estas "características comunes"bajo los nombres **"de algo"**, **comprensión**, **representación/abstracción** y **omisión**, características que vamos a analizar con algo más de detalle.

Algo indiscutible en cualquier definición de modelo es que un modelo es siempre un modelo **de algo**. Este "algo" recibe diferentes nombres en la literatura: sistema,

organización, dominio de aplicación, dominio de negocio, universo de discurso... La delimitación de este algo, es decir, el decidir dónde empieza y termina aquello que se quiere modelizar es una tarea compleja, puesto que en muchas ocasiones la frontera entre lo que se quiere modelizar y lo que no, es difusa e incluso cambiante en el tiempo. Esta tarea está relacionada en gran medida con la disciplina, englobada dentro de la Ingeniería del Software, de la *Ingeniería de Requisitos*, que trata sobre todo aquello relativo a las condiciones que queremos que cumpla el modelo que vamos a construir. Hagamos notar que hasta ahora no hemos hecho mención de forma explícita a la "realidad" o al "mundo real". Si bien es cierto que en muchas ocasiones la modelización de sistemas de información tiene que hacer referencia a la realidad, es decir, lo que se modeliza son aspectos del mundo real, físico, cotidiano, no todos los modelos tienen esta característica. No hay más que pensar en los modelos que se realizan para diseñar objetos que todavía no existen (como por ejemplo los planos de un edificio, o los diseños para un nuevo coche), o los que se realizan para crear programas de ordenador, que a su vez pueden manipular construcciones virtuales, tales como entidades u operaciones matemáticas (grupos, espacios vectoriales, integrales, etc.). Adelantando un poco uno de los puntos clave que trataremos después, también podemos construir modelos cuyo dominio, cuyo ámbito, sean a su vez otros modelos.

Pero, ¿cuál es el objetivo de (la creación de) un modelo? ¿Para qué se crea y utiliza un modelo? De las definiciones anteriores se deduce que uno de los objetivos principales de la creación de un modelo es la **comprensión** del dominio que se está modelizando. En general, este dominio será amplio y complejo, y la mente humana no es capaz de capturar de manera simultánea todos los aspectos de tal dominio. De esta forma, a través del modelo se intenta una mejor comprensión de dicho dominio. Relacionado con este objetivo se encuentra el de la comunicación. Aunque habrá casos en que así sea, habitualmente no se construye un modelo para un uso estrictamente personal, es decir, para el uso exclusivo de la persona que construye el modelo. Antes al contrario, un modelo habitualmente es utilizado como medio para compartir conocimientos e información sobre el dominio modelizado, y también muy frecuentemente los modelos son realizados por un equipo de personas, que a su vez pueden querer comunicarse con otro grupo distinto. Trataremos un poco más adelante de la relación entre modelos, comunicación y lenguajes.

Las otras ideas básicas que subyacen a las definiciones precedentes están relacio-

nadas en esencia con el "cómo" se crea un modelo. En estas definiciones se introduce que un modelo es una **representación** y/o una **abstracción** de aquello que se quiere modelar. La idea de representación es la idea de *sustitución*, es decir, que si un modelo es una representación, es porque sustituye o reemplaza, en un determinado contexto, a aquello que se modeliza. De esta forma, un modelo, como representación de algo, nos tiene que conducir, si bien en general de manera indirecta, a ese algo. Por su parte, abstraer significa *considerar la esencia de las cosas*, lo que en particular puede realizarse aislando, separando las cualidades o propiedades fundamentales de cada cosa. Estas operaciones están profundamente relacionadas con el hecho de que un modelo necesariamente **omite** aspectos de aquello que se modeliza. Esto fundamentalmente es debido a que, por una parte, lo que se quiere modelizar es habitualmente complejo, y por tanto el modelo no puede abarcarlo en su totalidad, y por otro porque en el intento de quedarse sólo con lo esencial (abstracción), el modelo necesariamente tiene que obviar ciertos aspectos del dominio modelizado.

Hasta aquí nos hemos limitado a analizar algunas características de la noción de modelo, que, aunque extraídas a partir de cuatro definiciones particulares, están ampliamente admitidas en la literatura, de forma que se corresponden con la idea más general de modelo. Vamos a introducir ahora nuestro propio punto de vista sobre este tema, poniendo nuestras ideas en relación con la noción habitual de modelo. Como cuestión previa, y bajo la perspectiva que adoptamos en este trabajo, *renunciamos de forma explícita e intencionada a proporcionar ninguna definición.* Esto es así porque entendemos que una *definición* debe ser algo riguroso y no ambiguo, y por tanto debe tener una mínima base formal. Las definiciones habituales en Matemáticas cumplen este requisito. Sin embargo, en el ámbito de la modelización de sistemas de información es difícil cumplir esto de forma estricta, ya que el método de trabajo incluye trabajar con conceptos, ideas, percepciones, opiniones, experiencias, experimentos, que en muchísimas ocasiones tendrán una fuerte componente subjetiva. Por tanto, y en particular, no nos planteamos como objetivo dar una definición de modelo, sino que, como ya adelantábamos en el título del apartado, intentaremos dar algunas ideas sobre esta noción. Es necesario hacer notar también que muchas de estas y otras ideas del presente trabajo tienen su base e inspiración en la incipiente disciplina de la *Fenomática* [Dom99].

En primer lugar debemos partir de que modelizar es una acción que realiza un sujeto humano, y que por tanto está no ya influenciada, sino directamente dirigida,

por la *percepción* que de las cosas tenga dicho sujeto. Nuestra postura filosófica es por tanto netamente **constructivista**: en este sentido nos identificamos plenamente con la distinción entre el punto de vista *objetivista* y el *constructivista* tal y como se muestra en el Informe FRISCO [FHL+98]:

- Objetivista (realista *naif*): alguien que cree que la "realidad" existe con independencia de cualquier observador y que simplemente necesita ser proyectado sobre descripciones adecuadas; para el objetivista, la relación entre la "realidad" y un modelo de dicha "realidad" es obvia o trivial.
- Constructivista: alguien que también cree que la "realidad" existe con independencia de cualquier observador, pero que es consciente del hecho de que sólo tenemos acceso a nuestras propias "concepciones" (mentales); para el constructivista, la relación entre realidad y concepción es principalmente subjetiva, y puede estar sujeta a negociación entre los observadores; cualquier acuerdo -lo que llamaremos "realidad inter-subjetiva"- puede ser revisado en cualquier momento a lo largo del tiempo.

Dicho de otro modo, cuando se intenta crear un modelo de algo, el modelo final que se obtiene no es en realidad un modelo directo de ese algo, *sino un modelo de la percepción* que el sujeto tiene de ese algo. Este punto, aunque pudiera parecer poco importante, está en la base de por qué se realizan malas interpretaciones sobre modelos concretos. Es claro que sobre un mismo dominio, dos sujetos distintos pueden tener (y en la mayor parte de los casos así sera) percepciones distintas de dicho dominio, lo que puede conducir a que ambos sujetos construyan modelos distintos. Observemos que admitimos como posible que a pesar de que las percepciones sean distintas, los modelos construidos sí que puedan ser iguales, o al menos, equivalentes [1]. Es evidente que esto se encuentra relacionado con los objetivos de un modelo, entre los que antes citábamos la *comprensión* y la *comunicación*. Cuando se trata de construir un modelo para comprender mejor un cierto dominio, debemos tener en cuenta la *componente subjetiva*, considerada no sólo como aquella interpretación

[1] Aunque sin lugar a dudas sería deseable, no parece adecuado profundizar en este momento sobre el tremendamente complejo problema de la igualdad: ¿qué quiere decir que dos modelos son iguales? ¿Qué quiere decir que sean equivalentes?

voluntaria y consciente que la persona realiza del dominio, sino la otra que estamos haciendo notar, que es la de la percepción que del dominio tenga la persona que construye el modelo. En el momento que este modelo deba ser comunicado, deba ser compartido por un grupo de personas, debemos tener en cuenta la *componente inter-subjetiva*, entendida no sólo como la suma de las componentes subjetivas de cada persona individual, sino lo que es más importante, como el acuerdo, ya sea tácito o explícito, que el grupo de personas conviene en adoptar para la interpretación de la realidad: en lenguaje coloquial, lo que se persigue es que todo el grupo "hable de lo mismo".

Partiendo pues de la percepción de algo que tiene la persona que trata de crear un modelo, dicho modelo será, en todos los casos, una *descripción (más o menos detallada) de dicha percepción*. Esta postura establece una diferencia clara respecto a lo que es habitualmente considerado en la literatura, y en particular respecto a las definiciones de ejemplo que hemos presentado. En estas definiciones, un modelo siempre era o bien una representación (y por tanto una sustitución) o bien (o además) una abstracción (es decir, una obtención de lo fundamental). Entendemos, que, por una parte, un modelo no ha de ser necesariamente una representación, sino que un modelo puede crearse con la intención de que sustituya a aquello que se modeliza o puede no crearse con esa intención. Es decir, el servir como representación puede ser uno de los objetivos del modelo, pero no es necesariamente el único. Por otra parte, el hecho de entender un modelo como una abstracción está más relacionado con el hecho de que el modelo se realiza sobre lo percibido que con la noción intrínseca de modelo. El acto cognitivo de la percepción va acompañado, en la mayor parte de los casos de forma involuntaria y/o inconsciente, de otros, tales como cuantificación, cualificación, clasificación, y en particular, abstracción. Por ejemplo, al percibir una mesa, de manera inconsciente veo que es una y sólo una mesa, que es de un determinado color, forma, y, ante todo, que en efecto es una mesa, y no cualquier otra cosa, con lo cual en efecto estoy realizando un proceso de abstracción que me permite identificar tal objeto como mesa. Cuando se crea un modelo, el objetivo del mismo puede ser el de realizar una abstracción de aquello que se modela, pero de nuevo esta es sólo una posible finalidad. Así pues, entendemos un modelo como una forma de describir aquello que se modeliza, considerando también cuáles son los objetivos se persiguen con su construcción.

Por último, un modelo es algo *necesariamente incompleto*. Las definiciones que

hemos tomado como ejemplo recogen la omisión de ciertos aspectos de aquello que se modeliza (con el objetivo, por ejemplo, de destacar otros), pero parece entenderse que esta omisión es siempre voluntaria por parte del creador del modelo. En efecto puede existir la voluntad de omitir parte del dominio modelizado, pero aunque tal voluntad no existiera, el modelo nunca podría capturar la totalidad del dominio, y esto por distintas razones, todas ellas ya expuestas directa o indirectamente con anterioridad. Por una parte, el dominio puede ser algo difícil de delimitar, y por tanto más lo será el modelo: la imposibilidad de recoger todas las interrelaciones entre los elementos que se desean modelizar, y de éstos con aquellos que están "fuera del dominio" obligan en todos los casos a detener en algún momento el proceso de modelización. Por otra parte, debemos reiterar que el modelo se construye a partir de la percepción de algo por parte un sujeto. Dicha percepción puede ser parcial o incluso distorsionada, por lo que el modelo obtenido puede no corresponderse de manera completamente fiel con aquello que se modeliza.

Aunque debemos insistir en que no pretendemos proporcionar una definición (al menos no en el sentido formal) de modelo, a modo de resumen podríamos decir que un modelo es una descripción, necesariamente incompleta, de la percepción que un cierto sujeto tiene sobre algo, y de tal forma que dicha descripción es creada con una finalidad concreta.

1.3. Modelos, lenguajes y comunicación

En la sección anterior hemos visto que algunos de los propósitos intrínsecos a un modelo pueden ser la comunicación entre varias personas y el de comprensión del dominio. Aún en este último caso, cuando el modelo fuera creado con la intención de profundizar en la comprensión de aquello que se modeliza para un uso estrictamente personal (de la persona que crea el modelo), siempre, para crear tal modelo, se hace uso de un cierto **lenguaje**. Observemos que los lenguajes existen por la intencionalidad de comunicar: aparentemente, un ser aislado, sin posibilidad de comunicarse con ningún otro ser, no necesitaría de ningún lenguaje, si bien es cierto que no existe certeza científica en este sentido. Sin embargo sí parece claro que no puede existir comunicación si no existe un mínimo nivel de lenguaje común entre los comunicantes. Además, obviamente, debe existir la *intención de comunicar*. En este sentido debemos diferenciar entre por una parte la intención abstracta

del lenguaje (de cualquier lenguaje) que es la de comunicar; por otra la intención concreta de cada lenguaje particular, que tendrá un uso determinado orientado a un ámbito específico; y por último, la intención con la que un usuario de un lenguaje concreto crea una expresión concreta de dicho lenguaje.

Con respecto a las cuestiones que nos ocupan, cuando una persona crea un modelo *siempre utiliza, aunque sea de manera indirecta o incluso involuntaria, un lenguaje*. Esto es así básicamente por la existencia del lenguaje cotidiano, al que en el ámbito de la modelización de los sistemas de información habitualmente llamamos **lenguaje natural**. Sin lugar a dudas, el lenguaje natural, ya sea hablado o escrito, es el más utilizado en la vida diaria para expresar y comunicar (no es sin embargo el único: en una conversación cara a cara el lenguaje corporal juega un papel fundamental; las personas con discapacidades utilizan lenguajes específicos adaptados a su problema concreto, tales como el lenguaje de signos de los deficientes auditivos o el lenguaje Braille de los deficientes visuales, etc.). Es evidente que el lenguaje natural escrito es el único que hemos utilizado hasta ahora en esta memoria, y nos viene sirviendo para expresar ideas con la intención de comunicarlas a otras personas. Es evidente también que nos estamos expresando en un idioma concreto, el castellano, pero también podríamos haber utilizado para expresarnos el lenguaje natural en idioma inglés, francés o alemán. Cuando se decide utilizar uno de los idiomas en concreto, esta decisión está lógicamente condicionada por el hecho de que se pretende que, en efecto, la comunicación se establezca con éxito, es decir, se pretende que el destinatario de nuestro mensaje llegue a comprender lo expresado. Esta reflexión, que parece poco menos que trivial considerada dentro de la distinción entre distintos idiomas, resulta clave cuando se traslada al contexto de la modelización de sistemas de información. Antes de profundizar más en este aspecto, creemos conveniente incidir en la contraposición de los términos *definición* y *descripción* que hemos realizado en el apartado anterior, poniéndola en relación con el uso del lenguaje natural. Y es que precisamente cuando se usa lenguaje natural es imposible proporcionar definiciones, sino simplemente descripciones. Si consultamos el diccionario, definir es "fijar con claridad y exactitud la significación de una palabra". Observemos que estamos utilizando el diccionario de forma recursiva, porque buscamos en él la *definición de definir*, y es claro que esta recursividad es una recursividad mal fundada: en un diccionario siempre obtendremos "definiciones circulares", ya que la "definición" de una palabra nos lleva a la definición de otra, y así indefinidamente. Esta discusión

no es nueva; Toril Moi cita en [Moi88] a Jacques Derrida (considerado el padre de la 'teoría' –mezcla de filosofía, literatura, etc- deconstructivista) el cual afirma que "el significado no está nunca presente, sino que está construido mediante el proceso potencialmente interminable de aludir a todos los restantes significantes ausentes. Se puede decir que el significante 'siguiente' da sentido al 'anterior', y así sucesivamente *ad infinitum*". En la misma obra, Toril Moi cita a Christopher Norris, el cual define la escritura como "*desplazamiento* interminable del significado que gobierna el lenguaje, situándolo para siempre fuera del alcance de un conocimiento estable y autentificador". Sin duda de nuevo estamos comenzando a trasladar nuestra discusión hacia los mundos de la psicología, la filosofía, la cognición o el aprendizaje, puesto que haría falta saber en qué momento una persona, a partir de una definición de diccionario, queda convencida de haber comprendido el significado de lo definido. Insistimos pues en que entendemos que en lenguaje natural sólo se proporcionan descripciones, nunca definiciones, que deberían reservarse al ámbito de lo formal[2].

Como ya hemos comentado, siempre que se crea un modelo, se utiliza un lenguaje. En el ámbito de los sistemas de información, la elección del lenguaje que se utiliza durante el proceso de modelización es un elemento crítico, ya que se trata de obtener un lenguaje que sea compartido -y por tanto comprendido- por todas las personas implicadas en el proceso de modelización, pero que además tenga la suficiente expresividad para ser capaz de recoger todos los aspectos, en muchos casos técnicos, necesarios para desarrollar el modelo. Lo que ocurre en la práctica es que un modelo no es *un* modelo, sino *un conjunto* de modelos, de forma que cada uno de ellos describe ciertas características de aquello que se modeliza, y con un cierto nivel de detalle: un modelo puede ser un *refinamiento* de otro, y este a su vez un *complemento* de un tercero. Y de forma pareja, no se utiliza un único lenguaje, sino un conjunto de lenguajes. Parece pues claro que el uso del lenguaje (de cualquier lenguaje, en general) y el de los distintos lenguajes (particulares de esta disciplina concreta) son claves en el desarrollo de los sistemas de información. Por tanto, en la siguiente sección intentaremos analizar qué es un lenguaje, particularizando qué es un lenguaje dentro de los sistemas de información, y viendo diferentes tipos

[2]Esta postura marca una diferencia con respecto a lo que habitualmente se encuentra en la literatura. En particular observemos que en el apartado anterior hemos puesto ejemplos de "definiciones"de modelo. Las hemos considerado como tales puesto que sus autores lo hacen así, pero a nuestro juicio deberían ser consideradas únicamente descripciones de la noción de modelo.

y clasificaciones posibles para los lenguajes dentro de este ámbito.

1.4. Una(s) idea(s) de lenguaje

De modo análogo a la sección *Una(s) idea(s) de modelo*, en esta sección vamos a tratar simplemente de aclarar algunas ideas sobre la noción de lenguaje, sin pretender realizar una revisión exhaustiva sobre las diferentes nociones y tipos de lenguajes que se han desarrollado.

Como ya hemos comentado en la sección anterior, los lenguajes aparecen unidos de manera indisociable a la idea de comunicación: sin lenguaje no puede existir comunicación. Ahora bien, ¿qué elementos proporcionan los lenguajes para establecer dicha comunicación? En esencia, lo que un lenguaje proporciona es un *conjunto de elementos atómicos* junto con una *serie de reglas* que permiten obtener *expresiones sintácticamente correctas* (también llamadas *expresiones bien formadas*). Los elementos atómicos se entienden como tales porque son las unidades del lenguaje más pequeñas que tienen sentido léxico, o dicho de otra forma, una división de cualquier elemento atómico no tendría sentido, no tendría significado en dicho lenguaje, pasando a ser un mero símbolo. Los elementos atómicos, combinados de forma que se respeten las reglas sintácticas del lenguaje permiten construir expresiones con significado no elemental. Como no puede ser de otra manera, para hablar acerca de los elementos que proporciona un lenguaje para expresar, es decir, al hablar de las *expresiones*, nos hemos visto obligados a hablar de sus *significados*. Dar un lenguaje es dar las expresiones permitidas dentro de ese lenguaje, pero obligatoriamente también qué significan, qué quieren decir, qué expresan (valga la redundancia) esas expresiones. Sin ser una terminología exclusiva del ámbito informático, es habitual hacer referencia a esta situación, de un modo algo más técnico, diciendo que para dar un lenguaje es necesario dar de forma separada, aunque evidentemente relacionada, su *sintaxis* y su *semántica*[3]. Insistimos en que las nociones de sintaxis y semántica no aparecen exclusivamente en el caso de lenguajes relacionados con la Informática, pero no es nuestro propósito dar ideas 'universales' sobre ellas, sino situarlas dentro del marco de la modelización de sistemas de información (en este sentido, una refe-

[3]En general, para proporcionar un lenguaje de forma completa no sólo es necesario dar su sintaxis y su semántica sino también su *pragmática*, es decir, cómo -dónde- debe ser utilizado dicho lenguaje, aunque no vamos a detenernos en este punto.

rencia sumamente ilustrativa, aunque centrada en un tipo concreto de lenguajes -los lenguajes de modelización- es el technical report de Harel y Rumpe [HR00]). Vamos pues a profundizar un poco más en este asunto.

En general, la sintaxis de un lenguaje comprende, por una parte el conjunto de conceptos básicos de dicho lenguaje, junto con las reglas que permiten describir conceptos más complejos. A esta parte se la suele denominar *sintaxis abstracta*, y casi siempre aparece acompañada de una *sintaxis concreta* o *notación*, es decir, de un conjunto de símbolos más o menos complejos que sirven para representar de forma explícita cada uno de los conceptos básicos. Volveremos de inmediato sobre este tema cuando tratemos la diferencia entre lenguajes textuales y lenguajes visuales/diagramáticos. Por otro lado, para proporcionar la semántica de un lenguaje lo más habitual es proporcionar, por una parte un *dominio semántico* y por otra una *función semántica*. El dominio semántico es habitualmente otro lenguaje "bien conocido", y la función semántica consiste en realizar una aplicación de las expresiones del lenguaje del que se está dando la semántica sobre expresiones de ese otro lenguaje "bien conocido". Ahora bien, ¿qué elecciones tenemos para este dominio semántico? O dicho de otro modo, ¿qué quiere decir disponer de un lenguaje "bien conocido"? Es bastante claro que nos encontramos con una situación análoga a la ya comentada para un diccionario y la cita de Derrida. Dar la semántica de un lenguaje es una acción pospositiva, puesto que se pospone la asignación de significados para la expresiones del lenguaje. Dicha asignación se realiza de manera indirecta a través de los significados de las expresiones de otro lenguaje (dominio semántico). Esta situación conduciría hipotéticamente a una cadena infinita de lenguajes de forma que para obtener la semántica de un lenguaje nos apoyaríamos en un segundo lenguaje, para obtener la de este segundo en un tercero y así indefinidamente. Sin embargo esta es una hipótesis planteada desde un punto de vista bastante extremo. En la práctica los lenguajes no se dan de este modo, sino que casi siempre se hace uso, de una forma u otra, del lenguaje natural. Aunque la sintaxis de los lenguajes (al menos una notación, una sintaxis concreta) si se suele dar de forma explícita, habitualmente la semántica se expresa de manera informal utilizando lenguaje natural. Puesto que la semántica del lenguaje natural se sobreentiende, esta forma de actuar termina la hipotética cadena de lenguajes a la que hacíamos referencia anteriormente. Aparentemente sólo existe otra forma de terminar la "cadena semántica" de lenguajes, que es llegar a un simbolismo absoluto, es decir, intentar expresar todo

a través de símbolos[4].

Es necesario hacer notar que cuando decimos que la semántica se expresa de manera informal, debido a la utilización del lenguaje natural, no necesariamente se debe inferir que por este motivo que la semántica tenga que ser obligatoriamente imprecisa. De nuevo la terminología es habitualmente confusa en la literatura, y existe una identificación como mínimo implícita de, por una parte, lo formal con lo simbólico, con lo riguroso, con lo preciso y otra identificación de lo informal con lo impreciso, lo ambiguo. Un tratamiento formal es un tratamiento simbólico en el sentido de *omisión de significado*, es decir, en el sentido de que el objeto del tratamiento formal queda desprovisto de significado, y sobre él se efectúa una manipulación simbólica[5]. Por tanto, lo formal, en sentido estricto, esta más relacionado con sintaxis que con semántica, y así, cuando se habla de *semántica formal* de un lenguaje, habitualmente se está haciendo referencia al hecho de que la semántica del lenguaje se define utilizando como dominio semántico un dominio simbólico, y por tanto, en cierta medida, carente de significado. También es habitual ligar el término 'formal' a las Matemáticas (por ejemplo, la disciplina informática de la *especificación formal* es definida en [Nis99] como la "aplicación de las matemáticas para la especificación de software, es decir, para documentar los requisitos del cliente sobre el software"). Esta asociación es coherente con el hecho de que la mayor parte de las Matemáticas también se ha construido apelando a un tratamiento simbólico, pero no podemos despreciar el hecho de que las Matemáticas también tienen una fuerte componente 'no-formal'.

Insistimos en que la utilización de lenguaje natural no significa obligatoriamente que el tratamiento sea impreciso o ambiguo, tal y como se identifica habitualmente. Es cierto que por su propia naturaleza el lenguaje natural contiene ambigüedades, provenientes de polisemias, sinonimias, homonimias, etc. Sin embargo entendemos que es posible utilizar el lenguaje natural de forma sistemática y rigurosa: una prueba

[4]Una empresa similar a ésta fue la emprendida por Russell y Whitehead en el tratado *Principia Mathematica* [WR10], que trataba de formalizar y axiomatizar todas las Matemáticas en un único sistema mediante el uso de conceptos lógicos formales. Este sistema no solo pareció tener un escaso aprovechamiento práctico, sino que resultó ser directamente inconsistente, tal y como fue demostrado por Gödel [Göd31].

[5]Observemos por otra parte que la otra connotación posible de la idea de *símbolo*, que es la de algo que refiere o recuerda a otra cosa, es en cierto modo opuesta a la que tiene que ver con los tratamientos formales.

de ello es la noción de Sistema de Referencia que mostramos en el capítulo dedicado a la presentación de la técnica de metamodelización NÓESIS. Debemos remarcar que no estamos diciendo que el tratamiento formal no sea necesario, y de hecho entendemos que una interesante línea de investigación en Informática consiste en la indagación de la obligatoria compatibilidad entre lo formal y lo informal. Esta compatibilidad debe buscarse de manera obligatoria porque en Informática existen dos actores principales, la máquina y el hombre, cuyos lenguajes (en sentido amplio) respectivos son prácticamente opuestos: el lenguaje propio de la máquina es el puramente formal, totalmente simbólico, desprovisto de significado, sin semántica intrínseca; el lenguaje propio del hombre es el puramente informal, cargado de significados y aún peor, de dobles sentidos. La Informática, como disciplina que se encuentra a mitad de camino entre la máquina y el hombre, no debería nunca perder de vista esta situación de partida. En el caso particular del tema de esta sección, el de los lenguajes en Informática y más en particular los lenguajes para la modelización de sistemas de información, una posible vía es la del estudio de lenguajes dotados de una doble semántica (los que podrían denominarse *lenguajes bisemánticos*), una netamente simbólica, orientada a la máquina, y otra más cercana al usuario humano.

1.4.1. Algunos tipos de lenguajes en Informática

Para concluir este apartado dedicado a los lenguajes, vamos a revisar algunas posibles clasificaciones de los mismos, basándonos en diferentes puntos de vista. Una primera clasificación atiende exclusivamente al tipo de sintaxis concreta, de notación, utilizada por los lenguajes. En este sentido podemos distinguir entre *lenguajes textuales* y *lenguajes diagramáticos*, también llamados *visuales*. Los lenguajes textuales son aquellos que hacen uso exclusivo de texto (secuencias lineales de símbolos alfanuméricos) en su notación, mientras que los lenguajes diagramáticos hacen uso de símbolos gráficos (además de texto) tales como cuadrados, rectángulos, curvas cerradas y arcos entre ellos, de forma que se explotan relaciones geométricas y topológicas tales como proximidad, conectividad, 'contenido', etc. entre los distintos elementos básicos para describir las expresiones permitidas del lenguaje (diagramas). Observemos que puesto que esta clasificación esta basada exclusivamente en el tipo de notación utilizada por cada lenguaje, nada impide la existencia de lenguajes que dispongan de dos sintaxis concretas diferentes, una de tipo textual y otra de tipo

visual, si bien esta situación no es muy habitual.

Otra clasificación más relacionada con la intención de los lenguajes es la que distingue entre *lenguajes de programación* y *lenguajes de modelización*. Al poner esta nueva clasificación en relación con la anterior, resulta cuando menos llamativo el hecho de que la gran mayoría de los lenguajes de programación son lenguajes textuales, y la mayor parte de los lenguajes de modelización son lenguajes diagramáticos. Así mismo es curioso el comprobar que para una amplía mayoría de los lenguajes textuales se ha proporcionado una sintaxis y semántica tal y como describíamos anteriormente, situación que no se traslada para los lenguajes diagramáticos, para los que en contadas excepciones podemos encontrar una descripción detallada y separada de su sintaxis y semántica (no al menos en el sentido formal -de provisión de función y dominio semánticos- al que hacíamos referencia con anterioridad).

La distinción entre lenguajes de programación y de modelización se encuentra bastante ligada a la (breve) historia de la Informática. El diseño de lenguajes de programación ya era tema de investigación para algunos de los padres de las Ciencias de la Computación, como Hoare [Hoa73] o Wirth [Wir74], mientras que el diseño de lenguajes de modelización es un tema de reciente actualidad (véase por ejemplo, [POB00]). De hecho los lenguajes de programación fueron los primeros lenguajes que tuvieron la denominación de tales, desde los primitivos lenguaje máquina y lenguaje ensamblador, hasta los lenguajes de alto nivel. La lista de estos últimos es interminable, y se corresponden con diferentes paradigmas de programación: Fortran, Basic, Cobol, Pascal, C, Lisp, Scheme, C++, Java... Con posterioridad a la aparición de los primeros lenguajes de programación comenzaron a surgir lenguajes de modelización, inicialmente ligados al desarrollo de la tecnología de bases de datos. Así aparecen el modelo jerárquico, el modelo en red, y, con una importancia mayor por su posterior universalización, el modelo relacional y el modelo entidad/relación (E/R)[6]. El modelo relacional provoca la consideración de diferentes tipos de lenguajes dentro del contexto de las bases de datos, tales como los lenguajes de definición de datos (DDL's, Data Definition Languages), los lenguajes de gestión de datos

[6]Tal y como estamos mostrando, es habitual referirse a estos lenguajes no como tales, sino como 'modelos'. Esta situación no es sino una muestra más de la confusión histórica existente al referirse a este tipo de nociones. Hoy en día parece evidente que, por ejemplo, el 'modelo' E/R debe ser considerado en realidad como un lenguaje que sirve para describir modelos. Sin embargo, y como mostraremos en seguida, el 'lenguaje' E/R también puede ser considerado como modelo, desde un nivel de abstracción diferente.

(DML's, Data Management Languages) o los lenguajes de consultas. Estos últimos establecen una peculiar categoría intermedia entre los lenguajes de programación y de modelización, puesto que aunque sirven principalmente para modelizar, admiten la inmersión de sentencias de lenguajes de programación, de forma que también pueden realizar tareas propias de estos últimos. Sin duda la referencia obligada entre los lenguajes de consultas a bases de datos relacionales es SQL, que aunque concebido inicialmente como tal lenguaje de consultas, incluye actualmente funcionalidades de DDL y de DML.

A partir del modelo E/R anteriormente citado surgen multitud de variantes, y casi simultáneamente van apareciendo un elevado número de lenguajes orientados a diferentes tareas de modelización, lenguajes tales como RM/T, Niam, ORM, OMT, Redes de Petri, Statecharts... sin olvidar UML, el lenguaje que ha sido recientemente adoptado como estándar para la modelización de sistemas software orientados a objeto. A partir de esta lista de lenguajes de modelización podemos extraer una nueva clasificación para este tipo de lenguajes. En el ámbito de las bases de datos el tipo de modelización que es necesario efectuar es esencialmente de tipo estructural, estático. Esto significa que el objeto de la modelización son sistemas para los cuales la mayor parte de los datos varían poco o nada a lo largo del tiempo. Por ejemplo, en una base de datos con información sobre personas la mayor parte de los datos para una persona dada no cambiarán nunca o en contadas ocasiones (DNI, nombre, apellidos, fecha de nacimiento), aunque sí habrá otros que podrán cambiar (dirección, número de teléfono) si bien con no mucha frecuencia. Sin embargo existen otros tipos de sistemas software en el que el elemento tiempo es esencial en el proceso de modelización, y la reacción del sistema a los cambiantes estímulos externos es fundamental. Ejemplos de este tipo de sistemas son los sistemas concurrentes o los sistemas reactivos, y dentro de estos los sistemas de tiempo real, como los utilizados para realizar tareas de regulación semafórica o de control aeronáutico. Es claro que el tipo de lenguaje que es necesario utilizar para modelizar una base de datos no puede ser el mismo que el necesario para modelizar un sistema de tiempo real. Así se distingue entre *lenguajes para la modelización de la estructura (estática)* y *lenguajes para la modelización del comportamiento (dinámico)* de los sistemas. Ejemplos de lenguajes para la modelización de la estructura (a los que llamaremos brevemente, y abusando en cierto modo del lenguaje natural, *lenguajes de estructura*) son los ya citados para la modelización de bases de datos como el modelo relacional, así como

E/R, Niam u ORM (a veces referidos también bajo la denominación de *lenguajes de modelización conceptual*). Por su parte, Redes de Petri y Statecharts son los casos más conocidos de lenguajes para la modelización de comportamiento (o *lenguajes de comportamiento*, cometiendo de nuevo un ligero abuso lingüístico). Algunos lenguajes modernos, tales como OMT y UML deberían considerarse cada uno de ellos como un conjunto de lenguajes, pues contienen sub-lenguajes que permiten realizar separadamente cada una de las tareas de modelización de estructura y comportamiento a las que hacíamos referencia. De hecho, la necesaria (y aun más, obligatoria) coordinación y compatibilidad que debe existir entre estos sub-lenguajes es tema de investigación constante, asunto que es especialmente notable y actual en el caso de UML.

Existen además otros tipos de lenguajes, que se encuentran de alguna manera a medio camino entre la programación y la modelización. Por ejemplo, los *lenguajes de especificaciones*, tales como SDL, Promela, Z u Object-Z, o los que podemos agrupar bajo la denominación *lenguajes de descripción de páginas*, tales como Postscript o HTML, si bien los ámbitos de aplicación de estos dos últimos son bastante diferentes, orientado a la impresión en papel en el primer caso, y orientado a la elaboración de páginas web en el segundo. Conectado con este último ámbito y relacionado también con el de las bases de datos se encuentra el lenguaje XML.

La existencia de esta multitud de lenguajes, con ámbitos de aplicación inicialmente tan distintos, pero que en muchas ocasiones deben utilizarse conjuntamente y coordinadamente, sugiere la necesidad de la existencia de un marco de trabajo que permita manipular los propios lenguajes como objeto de modelización (el 'qué' se modeliza) de forma que los lenguajes se puedan analizar, adaptar y/o transformar. El estudio de esta situación será nuestro próximo objetivo.

1.5. Modelización de modelos y de lenguajes: metamodelización

Acabamos de mostrar en la sección anterior que en el ámbito de la Informática existen multitud de lenguajes, y que la propia noción de lenguaje es un concepto fundamental en este contexto. Esto ha provocado que recientemente se haya comenzado a considerar el estudio y modelización de los lenguajes como una disciplina

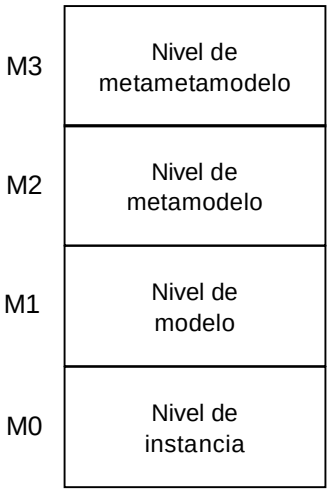


Figura 1.1: Arquitectura de cuatro niveles

en sí misma. En el proceso de modelización que podemos llamar "tradicional" se crea un modelo de algo que (en la mayor parte de los casos) pertenece al mundo real, con la intención de que ese algo pueda ser trasladado de manera fiel al ordenador. En la nueva situación el 'algo' que ha de ser modelizado es un lenguaje, que a su vez es utilizado como herramienta para crear otros modelos. En ocasiones esta situación es interpretada diciendo que el objeto de la modelización es el conjunto de modelos que se pueden crear con el lenguaje. Sea cual sea la interpretación elegida, el término **metamodelización**[7] se encuentra plenamente admitido en la literatura como referencia para esta situación (véanse, por ejemplo, [SLTM91, TL93, MPBE95, Ode95, GKP98, Bez98, Drb99, NSKL99]). Al considerar una perspectiva de metamodelización surgen modelos de distinta índole, y de manera natural aparece la necesidad de considerar diferentes niveles donde situar tales modelos.

En la literatura podemos encontrar diferentes terminologías que se corresponden básicamente con esta idea. En este apartado revisaremos una perspectiva relativamente sencilla, que es la considerada en la definición de UML. Esta perspectiva recibe el nombre de *Arquitectura de Cuatro Niveles* [OMG01b]. Dichos niveles aparecen representados en la figura 1.1. Intentaremos ilustrar esta arquitectura a través

[7]El prefijo *meta-* proviene del griego, y significa en castellano *más allá de, posterior a* . Por lo tanto "metamodelización" es aquello que está más allá de la modelización, que es posterior a la modelización.

de un ejemplo bastante clásico. Imaginemos que deseamos modelizar información acerca de los empleados de una empresa, y que para ello vamos a utilizar un lenguaje de modelización orientado a objeto, como por ejemplo UML. En el modelo que crearemos deberá aparecer con toda seguridad una clase 'Empleado', con atributos tales como 'nombre', 'fecha de nacimiento', etc, según se haya decidido en la fase de análisis cuáles son las características de los empleados que se desean recoger. Los objetos de la clase 'Empleado' modelizarán los individuos (empleados) concretos del mundo real. Con respecto a la Arquitectura de Cuatro Niveles, los objetos concretos de la clase se corresponden con el Nivel de Instancia, y el modelo creado (en particular la clase 'Empleado') se corresponde con el Nivel de Modelo. Observemos que en realidad hasta aquí simplemente hemos revisado un ejemplo de modelización 'tradicional'. La nueva visión consiste en considerar aquellos conceptos que se han utilizado para crear el modelo como elementos que a su vez son susceptibles de ser modelizados. Así, consideramos un nuevo Universo de Discurso en el que los individuos son los conceptos 'Clase', 'Objeto' o la relación o relaciones existentes entre ambos. Por tanto, este nuevo Universo de Discurso es el propio lenguaje de modelización que hemos utilizado, y el nuevo objetivo es crear un modelo de tal Universo de Discurso, es decir, crear un modelo del lenguaje. A este nuevo tipo de modelos se les denomina **metamodelos**, y dentro de la Arquitectura de Cuatro Niveles dichos modelos se encuadran en el Nivel de Metamodelo.

Vamos a detenernos en este punto a revisar como encaja esta nueva situación con las ideas de modelo y lenguaje que hemos intentado aclarar en las secciones anteriores. En primer lugar recordemos que habíamos descrito un modelo como una descripción de algo, necesariamente incompleta y creada con una cierta intención. ¿Un metamodelo es concordante con estas características? Dicho de otro modo, ¿un metamodelo es en particular un modelo? Sin duda un metamodelo es una descripción, en este caso de un lenguaje. Y precisamente ésta, el ser una descripción, es una de las intenciones por la que se crea un metamodelo. Habitualmente, el metamodelo se crea para aclarar el significado de un lenguaje, de forma que se expresen con claridad los conceptos fundamentales de dicho lenguaje. Dicho de otra forma, un metamodelo describe la sintaxis (abstracta) del lenguaje que se metamodeliza. Es difícil encontrar metamodelos que muestren además una notación (sintaxis concreta) o la semántica del lenguaje metamodelizado. Este punto ha sido destacado por varios autores, como por ejemplo en [KER99] en el caso particular del metamodelo *oficial*

de UML o en [BSH99, tHV97] dentro del contexto más general de la *Ingeniería del Método*. Esta situación es especialmente llamativa cuando el metamodelo se utiliza como definición del lenguaje. En este caso estamos usando el término 'definición' con el sentido formal de 'creación' de algo, y recordemos que para dar un lenguaje es necesario dar tanto su sintaxis como su semántica. Un metamodelo que se da con la intención de definir un lenguaje debería contener ambos aspectos. De nuevo el caso de UML es paradigmático en sentido negativo, puesto que los documentos de definición del lenguaje [OMG01b], aún a pesar de contener una parte titulada "Semántica de UML", recogen casi únicamente aspectos sintácticos del lenguaje, aspecto que también ha sido resaltado por los autores de [KER99]. Esta situación esta relacionada también con la característica de 'incompletitud' que hemos asociado con la noción de modelo. En el caso de que el metamodelo del lenguaje se cree con intención puramente descriptiva, es bastante probable e incluso lógico que el metamodelo sea incompleto. La situación es análoga a la que encontrábamos en el caso de modelos 'tradicionales': puesto que la realidad a modelizar es en general muy compleja, el modelo debe capturar sólo parte de dicha realidad, resaltando unos aspectos e ignorando otros. De la misma forma, cuando se trata de describir (modelizar) un lenguaje será habitual crear modelos incompletos, en función de la complejidad del lenguaje. En contraste, si el modelo se desarrolla con la intención de definir el lenguaje, éste obviamente debe ser completo, o dicho de otra forma, el metamodelo debería definir el lenguaje en su totalidad. Más adelante, en la sección dedicada a la ingeniería del método, se mostrarán otras intenciones de uso posibles de un metamodelo.

Parece pues claro que en efecto todo metamodelo es, en particular, un modelo. Esto nos conduce a un nuevo interrogante. Según hemos visto, para expresar cualquier modelo se utiliza al menos un lenguaje (el lenguaje natural, en el caso más extremo). Entonces, ¿qué lenguaje o lenguajes se utilizan para describir metamodelos? La literatura proporciona básicamente dos respuestas a la pregunta anterior. La primera opción, a la que podríamos denominar *definición circular*, consiste en utilizar, para expresar el metamodelo de un lenguaje A, el propio lenguaje A[8]. Este

[8]Como curiosidad, en [CEF+99] se sugiere que quizá el término 'metamodelización' debería reservarse para esta situación, en el que el metamodelo de un lenguaje se describe utilizando (un subconjunto de) el propio lenguaje. Esta no es, sin embargo, la opción utilizada generalmente en la literatura, en la que el término 'metamodelo' se utiliza cuando lo modelizado es un lenguaje.

es (aparentemente) el enfoque adoptado por UML, puesto que en efecto el metamodelo de UML está expresado en UML. Y decimos aparentemente porque en la descripción de la Arquitectura de Cuatro Niveles se dice textualmente que "el metamodelo de UML es una instancia del meta-metamodelo de MOF", de lo que parece deducirse que el metamodelo de UML está escrito en realidad utilizando el lenguaje MOF (Meta Object Facility, [OMG01a]). Sin embargo no acaba de estar clara la relación entre UML y MOF (¿Es MOF un sub-lenguaje de UML? ¿O es un lenguaje diferente que comparte –como mínimo– la notación con UML?). En cualquier caso, en efecto el cuarto nivel de la Arquitectura de Cuatro Niveles (que no habíamos descrito hasta ahora) se corresponde con MOF en el caso de UML, y más en general con el lenguaje utilizado para expresar los metamodelos. Esto nos conduce hasta la segunda elección que podemos encontrar en la literatura para los lenguajes de metamodelización, y que es la de desarrollar lenguajes específicos para esta tarea. De nuevo la literatura es bastante extensa en lo referente a este ámbito, y así podemos encontrar distintos lenguajes de metamodelización por ejemplo en [HSB97], [KLR96], [MBJK90], [Sae95], [VtH95], [CEKS01], [DZR97], e incluso la comparación de varios lenguajes en [HS96].

A modo de resumen, podemos describir un metamodelo como un modelo que, ya que es modelo de un lenguaje, refiere a una familia de modelos (todos los modelos que se puedan describir con dicho lenguaje). Como modelo que es, un metamodelo se expresa utilizando un lenguaje (de metamodelización).

1.5.1. Niveles de modelización y metamodelización

El nivel de abstracción a que conducen las cuestiones que estamos considerando en esta sección hacen que sea muy fácil perderse en la maraña de modelos, metamodelos, lenguajes, etc, así que no está de más recapitular la situación que tenemos en este momento. Partimos de una cierta realidad –dominio, Universo de Discurso– que es necesario modelizar (sin importarnos en este preciso momento cuál es la razón de tal necesidad). Para crear los modelos disponemos de una gran variedad de lenguajes. Cada lenguaje, a su vez, puede constituir un nuevo Universo de Discurso, susceptible de ser modelizado, o con mayor propiedad, metamodelizado. Y de nuevo, para crear los (meta)modelos disponemos de una variedad de lenguajes. ¿Qué tienen de diferente una fase y otra? Es decir, ¿qué diferencias existen entre los lenguajes de

modelización y los lenguajes de metamodelización? O dicho de otro modo, ¿ocupan unos y otros niveles de abstracción diferentes o nos encontramos en el mismo nivel, y lo único que cambia es el escenario, el Universo de Discurso? Intentemos contestar a esta pregunta, revisando de nuevo la situación de partida. A partir de un cierto Universo de Discurso, un diseñador humano percibe ciertos entes que él conceptualiza, abstrae, clasifica, para crear un modelo mental de ese dominio. Planteada de forma tan simple, en esta situación sólo hay dos niveles, el de los entes que se perciben (que jugarían el rol de instancias) y el de la conceptualización que se hace de ellos (que jugarían el rol de modelo). Aparentemente esto sería así con independencia de cuál fuera el Universo de Discurso de partida, no importando si dicho Universo es un cierto dominio del mundo real, o si es un lenguaje de modelización. Sin embargo, lo que ocurre en realidad es que para realizar la conceptualización de aquello que el diseñador percibe, tal y como ya hemos visto, siempre se hace uso de algún lenguaje. De hecho, el lenguaje utilizado puede condicionar (y en la mayor parte de los casos así será) el modelo que se obtiene como resultado final, y esto aunque tal modelo sólo exista en la mente del diseñador. Esto se encuentra relacionado también con las guías metodológicas que los lenguajes proporcionan. Por ejemplo, un diseñador que haya trabajado habitualmente utilizando el modelo E/R, a la hora de crear un modelo, intentará identificar en primer lugar, e incluso de forma involuntaria, las entidades que se puedan distinguir en el Universo de Discurso. Por tanto sí parece razonable admitir la existencia de tres niveles, el nivel de instancias, el nivel del modelo, y el nivel del lenguaje que se utiliza para describir el modelo. Es necesario remarcar que es más que probable que un diseñador en realidad no se plantee la existencia de estos tres niveles, sino que considere el lenguaje que utiliza como algo en cierto modo 'externo' al proceso de modelización, como una herramienta más que juega un papel similar al que cumplen, salvando las distancias, el bolígrafo y el papel o el propio ordenador personal. Probablemente para el diseñador no "existen" más que el modelo que está creando y las instancias de ese modelo. Sin embargo, desde el punto de vista de la metamodelización es importante considerar los tres niveles. Este patrón sí se repite con independencia de cuál sea el Universo de Discurso de partida. Si el Universo de partida es una parte del mundo real, tenemos el lenguaje (correspondiente al nivel M2 en la Arquitectura de Cuatro Niveles, figura 1.1) que se utiliza para crear el modelo (correspondiente al nivel M1), el cual referirá a un conjunto de instancias (nivel M0). Si el Universo de partida es un lenguaje de mode-

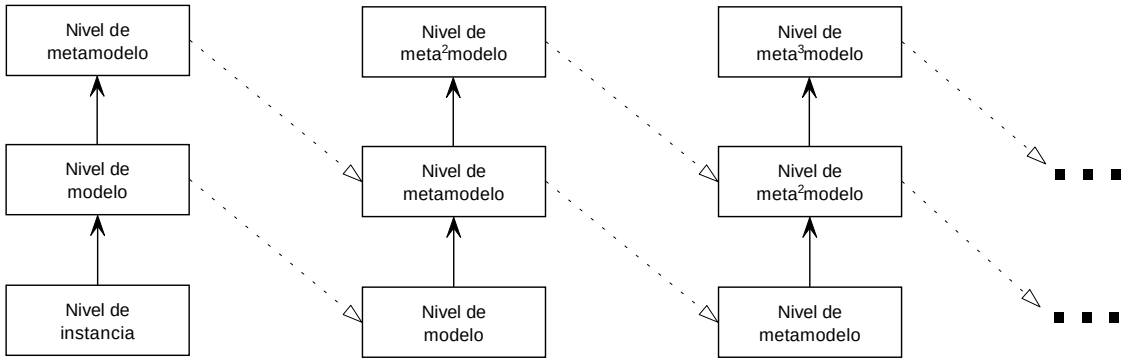


Figura 1.2: Arquitectura de metamodelización en zigzag

lización, tenemos el lenguaje (de metamodelización) (M2) que se utiliza para crear el (meta)modelo (M1), el cual referirá a un conjunto de instancias (que en este caso son los modelos que se pueden describir con el lenguaje de modelización de partida) (M0). Si el Universo de partida es un lenguaje de metamodelización, tenemos el lenguaje (que puede ser llamado de meta-metamodelización, meta^{2}modelización e incluso simplemente de metamodelización) (M2) que se utiliza para crear el meta-metamodelo (o meta^{2}modelo) (M1), el cual referirá a un conjunto de instancias (que en este caso son los metamodelos que se pueden describir con el lenguaje de meta-modelización de partida) (M0). Hipotéticamente este proceso podría dar lugar, una vez más, a una cadena infinita de niveles, modelos y lenguajes. De hecho, existen al menos dos propuestas de arquitecturas de metamodelización que plantean la existencia potencial de ese infinito. Una es la que hace uso del patrón Zigzag (ver figura 1.2) y otra es la arquitectura anidada propuesta en [AES01b] (ver figura 1.3). Una vez más, tal cadena no se llega a formar, en primer lugar porque en un momento u otro hay que hacer uso del lenguaje natural. Otra posibilidad para la no formación de la cadena es el uso de la definición circular: en un momento determinado del proceso se puede decidir que el metamodelo del lenguaje de meta^{n}modelización (donde n podría ser 0 –lenguaje de modelización–, 1 –lenguaje de metamodelización–, etc.) se describa utilizando ese mismo lenguaje de meta^{n}modelización. Esta puede ser la razón por la que en [AES01a] se admite que a pesar de que teóricamente se puede plantear la Arquitectura Anidada Genérica, "sólo las primeras representaciones de metanivel tienen algún valor real en la práctica", lo que recupera parcialmente la Arquitectura de Cuatro Niveles que hemos estado revisando.

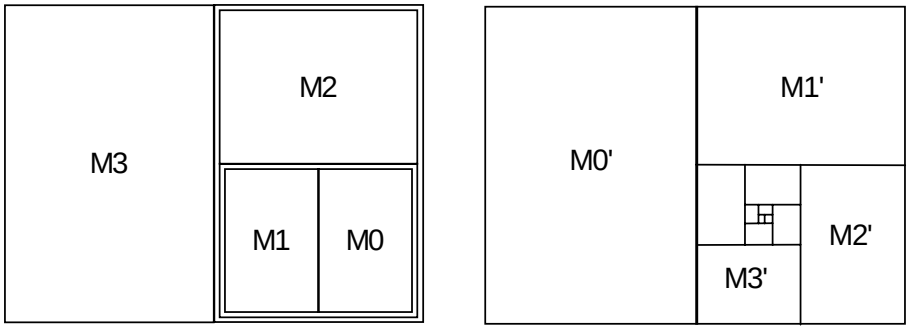


Figura 1.3: Arquitectura de metamodelización anidada

Por otra parte, la consideración de un patrón de tres niveles (instancia, modelo, lenguaje/metamodelo) es posiblemente más concordante con la práctica habitual tanto de la modelización como de la metamodelización. En efecto, como ya hemos observado, un analista (y/o diseñador) 'tradicional' debe conocer (al menos) un lenguaje para poder describir modelos, de tal manera que su interés principal es que las instancias de dichos modelos sean fieles con aquello que se está modelizando. Un analista 'tradicional' probablemente no necesitará conocer en profundidad cómo está definido el lenguaje que utiliza, o dicho de otro modo, no necesitará estudiar un metamodelo del lenguaje. Esto es claro en el caso de los lenguajes de programación: para aprender a crear programas (modelos) –de los cuales interesa su ejecución (instancia)– es necesario aprender un lenguaje de programación, pero no es necesario estudiar un metamodelo del lenguaje. Por su parte, cuando el enfoque es puramente de metamodelización, el analista (denominado habitualmente *meta–analista*) debe conocer (al menos) un lenguaje de metamodelización para poder describir metamodelos. El interés principal en este caso es que las instancias de dicho metamodelo sean fieles con los modelos que se pueden crear con el lenguaje que se está metamodelizando. Observemos que en este ámbito el meta–analista se encuentra sensiblemente distanciado de las instancias 'finales', es decir, de las instancias de los modelos que se describen con el lenguaje metamodelizado. Aunque también han de tenerse presentes, puesto que el fin del lenguaje metamodelizado es facilitar la modelización de tales instancias, no son en realidad un objetivo de primer orden para el meta–analista.

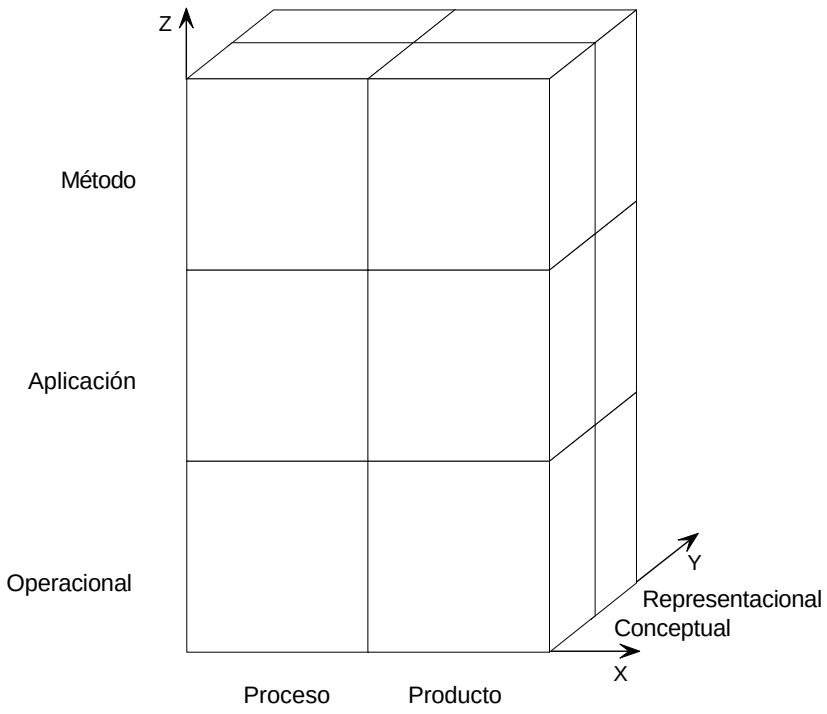


Figura 1.4: Dimensiones de modelización

Otra perspectiva que es interesante analizar al tratar sobre los niveles de modelización y metamodelización es la presentada por Hofstede y Verhoef en [tHV97]. Esta perspectiva se encuadra dentro del contexto de la Ingeniería del Método (la cual abordaremos de inmediato), y propone la consideración de tres dimensiones ortogonales para tratar con información sobre modelización. Estas dimensiones pueden ser representadas gráficamente en la forma de un cubo 3x2x2, tal y como se muestra en la figura 1.4[9], figura en la que hemos indicado explícitamente unos ejes cartesianos ficticios, que no aparecen en la referencia [tHV97], con el objetivo de facilitar la explicación de esta perspectiva. La dimensión del eje Z se corresponde con los llamados Niveles de Abstracción, y es equivalente a los tres niveles inferiores de la Arquitectura de Cuatro Niveles que hemos estado discutiendo. En concreto el Nivel Operacional del cubo de Hofstede y Verhoef se corresponde con el Nivel de Instancia (M0), el Nivel de Aplicación con el Nivel de Modelo (M1), y el Nivel de Método con el

[9]La figura 1.4 no es en realidad un cubo, pero este es el término –*cube*– utilizado por los autores en su documento original [tHV97].

Nivel de Metamodelo (M2). En principio, dentro de esta nueva perspectiva no existe un nivel que se corresponda con el Nivel de Meta-metamodelo (M3) de la Arquitectura de Cuatro Niveles. Sin embargo, el propio Hofstede junto con otros autores presentan en [WtHvO92] una versión previa a este cubo en el que aparece un nivel por encima del Nivel de Método, llamado Nivel Axiomático, que sí se correspondería con el Nivel de Meta-metamodelo. Por tanto el Nivel Axiomático es aquel que trata sobre los artefactos utilizados para producir los artefactos del Nivel del Método. A diferencia de la Arquitectura de Cuatro Niveles, que está estructurada linealmente, en una única dimensión, el cubo de Hofstede y Verhoef incluye dos dimensiones más. La dimensión del eje Y en el cubo establece la diferencia entre la modelización de información a un nivel conceptual y la modelización a un nivel de representación. Los autores explican claramente esta dicotomía: "Evidentemente, los modelos deben ser representados de una forma u otra: diagramas, matrices, tablas, listas, y especificaciones de programas son ejemplos. Debe establecerse una clara distinción entre los *conceptos de modelización* y su *notación externa*". Esta distinción está relacionada con la separación entre sintaxis abstracta (e incluso semántica) por un lado y sintaxis concreta (notación) por otro, que hemos discutido en secciones anteriores. De hecho, nuestra discusión se ha centrado en el nivel del lenguaje, al aclarar que para dar un lenguaje es necesario dar tanto sintaxis abstracta como concreta, acompañadas de semántica. La separación en el nivel del lenguaje se alinea con el Nivel de Método en el cubo, pero es fácil ver que la diferencia entre conceptos y notación también aparece en los niveles de Aplicación y Operacional. Por ejemplo, en el caso de un modelo concreto (Nivel de Aplicación) aparecerán los conceptos relacionados con el Dominio que se modeliza, que se instanciarán sobre conceptos o valores concretos en el Nivel Operacional. Todos estos conceptos se representarán (notación) de una forma u otra, representación que estará probablemente en función del lenguaje (y por tanto la sintaxis concreta) que se haya decidido usar. Por último, la dimensión en el eje X en el cubo de Hofstede y Verhoef es la que establece la existencia de los *productos de la modelización* frente a los *procesos de la modelización* [10]. En otras palabras, esta dimensión establece que es necesario distinguir entre el "qué" va a ser

[10]Los aspectos 'producto' y 'proceso' son también denominados, respectivamente, 'forma-de-modelizar' (way-of-modelling) y 'forma-de-trabajar' (way-of-working) por otros autores (véase por ejemplo, [Wij91, RSM95]), aunque probablemente esta nomenclatura sea algo más confusa que la utilizada por Hofstede y Verhoef.

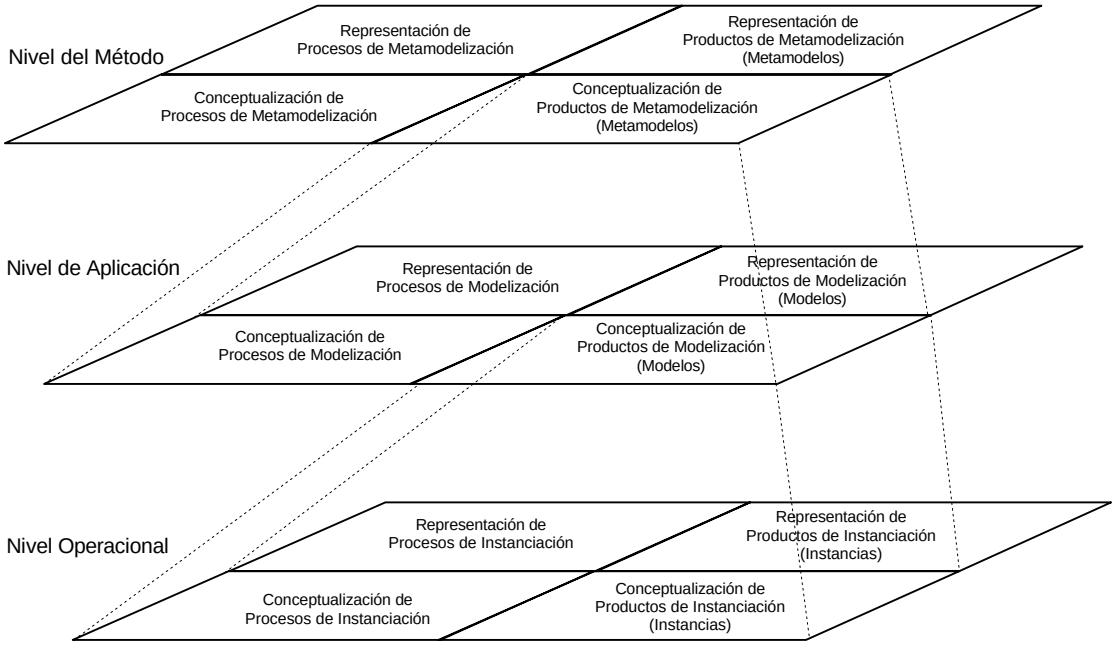


Figura 1.5: Interpretación de dimensiones de modelización

producido y manipulado y el "cómo" va a ser producido y manipulado ese "qué". Por ejemplo, en el Nivel de Aplicación, se distingue entre el modelo (producto de la modelización) y los pasos concretos (procesos de la modelización) que se han seguido para crear dicho modelo, tales como el análisis de requisitos o el refinamiento de cierta parte del modelo. En el Nivel de Método, los productos se corresponden, para cada lenguaje, con los conceptos de modelización que dicho lenguaje proponga (con independencia de si estos conceptos describen a su vez productos o procesos de modelización), lo cual constituye un metamodelo de dicho lenguaje. Los procesos en el Nivel del Método tratarían del cómo se crean dichos metamodelos.

En la figura 1.5 mostramos gráficamente nuestra interpretación del Cubo de Hofstede y Verhoef, la cual está inspirada también por la versión previa de este cubo que tal y como hemos comentado aparece en [WtHvO92]. Observemos que en esta interpretación consideramos, por ejemplo, que un metamodelo (es decir, una conceptualización de productos de metamodelización) debe *a priori* recoger, debe hablar, tanto del aspecto producto como del aspecto proceso en el nivel inferior, es decir, en el Nivel de Aplicación. En la representación gráfica en forma de cubo proporcionada por Hosftede y Verhoef esta situación no se deduce de manera clara,

ya que en el cubo podría interpretarse que un metamodelo (producto de metamodelización) debe hablar tan sólo de los productos de modelización, mientras que los procesos de metamodelización deberían tratar sobre los procesos de modelización. Entendemos que esta última interpretación no es correcta, y que es más adecuada la que mostramos en la figura 1.5. Observemos también en esta figura que en cualquier caso los aspectos de representación no se mezclan con los aspectos de conceptualización. Por ejemplo, un metamodelo no incluye aspectos de representación de los modelos sobre los que trata, sino que sólo abarca los aspectos conceptuales (tanto en productos como en procesos, insistimos). Tal y como hemos comentado anteriormente los modelos serán representados (notación) de una forma u otra, y aunque será conveniente acompañar al metamodelo de una notación concreta, el metamodelo en sí no tiene porqué contener estos aspectos. Este extremo es confirmado por ejemplo por la manera en la que se especifica el Lenguaje Unificado de Modelización (UML). En el Documento de Especificación del lenguaje UML [OMG01b] se muestran en capítulos separados el metamodelo de UML (bajo el epígrafe "Semántica de UML") y la notación de UML (bajo el epígrafe "Guía de Notación de UML").

1.5.2. Metamodelización avanzada: Ingeniería del Método

De lo expuesto hasta ahora, es fácil inferir que las ideas y prácticas relativas a la metamodelización tienen un alto grado de complejidad. Tanto es así, que dentro del ámbito general del desarrollo de sistemas de información están comenzando a surgir nuevas disciplinas que tienen a la metamodelización como una de sus herramientas fundamentales. Entre estas nuevas disciplinas hay que destacar, por su progresiva aceptación, la **ingeniería del método** [BLW96]. Como tal nueva disciplina, sus problemas y cuestiones abiertas van desde las mayores, relativas a su finalidad y contenido, hasta las aparentemente más nimias, como las terminológicas. Trataremos de mostrar algunas ideas tanto de unas como de otras, y precisamente comenzaremos intentando resolver algunas cuestiones de tipo terminológico.

La más básica de todas puede ser el por qué nos encontramos ante la ingeniería "del método" (utilizando el artículo determinado, 'el'), ya que parece natural plantearse la duda de a *qué* método nos referimos. El problema básicamente se presenta por la falta de literatura en lengua castellana sobre el tema, por una parte por la inexistencia de textos escritos originalmente en castellano, y por otro por la ausencia

de traducciones de artículos y libros del inglés. Aunque de inmediato proporcionaremos una descripción más precisa del término, al hablar de "ingeniería del método" no se hace referencia al tratamiento de *un* método en concreto, sino de *los* métodos, en general. Hemos decidido adoptar "ingeniería del método" como traducción más correcta del término anglosajón "method engineering", de la misma manera que se ha estandarizado en castellano el uso del término *ingeniería del software* como traducción del inglés *software engineering*.

Como ya hemos observado anteriormente, fijar la terminología que debe utilizarse al hablar de ingeniería del método no es un asunto en absoluto trivial. De hecho, el artículo [Bri96], en el que se propuso el término 'method engineering' como denominación, y que es uno de los artículos fundamentales para el establecimiento de esta disciplina como tal, está en buena parte dedicado a aclarar distintos términos relacionados con la ingeniería del método, términos que son utilizados en ocasiones de forma ambigua o confusa. Más en concreto, el autor de este artículo, Sjaak Brinkkemper, que puede ser considerado uno de los padres de la disciplina, escribe textualmente en el mismo:

> Durante décadas el mundo de los sistemas de información ha tenido problemas con su terminología. Esto es debido tanto a su juventud como a las influencias comerciales. Para establecer una base científica adecuada necesitamos ponernos de acuerdo en una buena terminología. Es esencial relacionar los términos con los de otras ramas de la ciencia que tengan enfoques de desarrollo metódico similares, tales como las ciencias organizativas o la ingeniería mecánica. [11]

[11] El problema de la inexistencia de una terminología ampliamente aceptada dentro del área del desarrollo de sistemas de información es también la motivación fundamental para la presentación del informe FRISCO [FHL+98], ya citado anteriormente en este mismo capítulo introductorio. La lectura de (como mínimo) el preámbulo y la introducción de dicho informe proporciona un diagnóstico muy acertado de la situación, y bastante coincidente con el citado de Brinkkemper. Por ejemplo, en el preámbulo se hace referencia al manifiesto FRISCO (del año 1988, y por tanto previo al informe), en el que se decía que:

> Existe una creciente inquietud dentro del *IFIP WG 8.1* [Grupo de trabajo 8.1 de la International Federation for Information Processing, dedicado al Diseño y Evaluación de Sistemas de Información] acerca de la situación actual, en la que están siendo usados demasiados conceptos difusos o mal definidos dentro del área de sistemas de información. Tanto la comunicación científica como la relacionada con el uso práctico están

Sin duda es cierto que dentro del área de sistemas de información existen una serie de términos que son usados en ocasiones como si fuesen poco menos que sinónimos. Así, se habla de técnicas, de métodos, de metodologías, de herramientas, de formalismos, de lenguajes... En las secciones anteriores de este capítulo, hemos intentado utilizar en todas las ocasiones el término 'lenguaje', que entendemos que es probablemente el más general de todos, pero existe una imperiosa necesidad de clarificar esta 'maraña terminológica'. De nuevo nos ayudaremos del artículo de Brinkkemper [Bri96]. Allí se definen (aunque recordemos que según nuestro punto de vista se debería decir "se describen") algunos de estos términos.

Un *método* es un "enfoque para llevar a cabo un proyecto de desarrollo de sistemas, basado en una forma específica de pensar, consistente en directrices y reglas, y estructurado de forma sistemática en actividades de desarrollo junto con sus correspondientes productos de desarrollo". Por tanto un método debe aportar una línea concreta de actuación, como por ejemplo el Análisis y Diseño Estructurado de SSADM, el enfoque de JSD o los distintos métodos que utilizan la orientación a objeto como idea central, tales como OMT, Objectory, o más recientemente Catalysis. Poniendo esta definición en relación con el Cubo de Hosftede y Verhoef analizado anteriormente, observemos que un método debe proporcionar tanto un aspecto 'proceso' (actividades de desarrollo, en la definición) como un aspecto 'producto' (productos de desarrollo).

Una *técnica* es, según Brinkkemper, un "procedimiento, posiblemente con una notación recomendada, para llevar a cabo una actividad de desarrollo". A la vista de

---

severamente distorsionadas y dificultadas, debido en parte a esta imprecisión, y debido también a la frecuente situación en la que diferentes interlocutores asocian diferentes significados a un mismo término. No existe una referencia conceptual y terminología comúnmente aceptada, que pueda ser aplicada a la definición o explicación de conceptos para los sistemas de información, ya sean nuevos o existentes.

A partir de este diagnóstico, el informe FRISCO se ofrece como "justificación científica para el área de los sistemas de información, colocándola en un contexto más general que abarca filosofía, ontología, semiótica, ciencias de los sistemas, ciencias organizativas, así como ciencias de la computación". Aunque este intento de comparar y contextualizar con otras disciplinas es común con el mostrado del artículo de Brinkkemper, el propósito del informe FRISCO es muchísimo más amplio, pues intenta cubrir todo el área de los sistemas de información. Puesto que esta sección está dedicada al sub-área de la ingeniería del método, fijaremos nuestra atención en el artículo de Brinkkemper, centrado de lleno en esta disciplina.

la definición de método, se deduce que un método concreto puede 'contener' distintas técnicas, ya que un método se estructura en diferentes actividades de desarrollo, y cada técnica particular está dedicada a una actividad de desarrollo. Ejemplos de técnicas son la modelización de datos con Diagramas E/R, la especificación de tareas con Casos de Uso, la modelización estructural con Diagramas de Clases o la especificación de comportamiento con Statecharts. Observemos que la definición de técnica no obliga a la existencia de una "notación" (por lo que, por ejemplo, la modelización de datos sería una técnica, con independencia de si se utilizan Diagramas E/R o Diagramas de Clases para realizar la tarea), pero lo habitual es que cada técnica vaya acompañada de una forma de expresión. A este respecto es necesario advertir que a pesar del esfuerzo realizado, Brinkkemper no es demasiado preciso en su definición de técnica. Como hemos explicado en secciones anteriores, una notación es simplemente la sintaxis concreta de un cierto lenguaje, y pueden existir distintas notaciones para un mismo lenguaje[12]. Sin embargo Brinkkemper aparentemente identifica los términos 'notación' y 'lenguaje', lo cual puede resultar confuso. Por tanto, a nuestro juicio una definición de técnica algo más precisa sería "procedimiento, posiblemente con un *lenguaje* recomendado, para llevar a cabo una actividad de desarrollo".

El siguiente término fijado en [Bri96] es *herramienta*, definido como "un medio posiblemente automatizado para respaldar una parte de un proceso de desarrollo". A la vista de la definición, prácticamente cualquier cosa (por ejemplo, el lápiz y el papel) podrían ser consideradas herramientas. En la práctica, con este término se identifican casi de manera exclusiva las herramientas de tipo informático (y por tanto automatizadas) tales como herramientas de Ingeniería del Software Asistida por Ordenador (Computer Aided Software Engineering-CASE) o Entornos Integrados de Soporte de Proyectos (Integrated Project Support Environments-IPSE).

Por último, Brinkkemper define *'la' metodología del desarrollo de sistemas de información* como la "descripción, explicación y evaluación sistemáticas de todos los aspectos del desarrollo metódico de sistemas de información". El autor remar-

[12]Utilizando un ejemplo propuesto también por Brinkkemper, para los Diagramas E/R existe un amplio consenso en expresar las entidades a través de cajas rectangulares, pero ya no hay una norma tan establecida para la expresión de las asociaciones (pudiendo utilizarse diamantes –lo más habitual, ya propuestos en el artículo original de P. Chen [Che76]–, o bien círculos –como por ejemplo en el modelo EAN de [Llo98]–).

ca que el frecuente uso erróneo del término 'metodología' cuando en realidad se está haciendo referencia a un 'método' es una prueba de la inmadurez del campo del desarrollo de sistemas de información. Para Brinkkemper, sólo existe *una* metodología del desarrollo de sistemas de información, si bien admite la existencia de algunas corrientes o 'escuelas' metodológicas diferentes, tales como la proveniente del mundo de la ingeniería del software o la originaria de la gestión de sistemas de información ligada al ámbito empresarial.

Una vez aclarada, al menos parcialmente, la terminología que debe utilizarse, ya es posible detallar una definición de *ingeniería del método*, descrita en [Bri96] como la "disciplina de ingeniería para diseñar, construir y adaptar métodos, técnicas y herramientas para el desarrollo de sistemas de información". La comparación con la ingeniería del software que hemos utilizado previamente es útil de nuevo para entender esta definición. De nuevo en palabras de Brinkkemper, "de la misma manera que la ingeniería del software se interesa por todos los aspectos de la producción de software, la ingeniería del método se ocupa de todas las actividades de ingeniería relacionadas con métodos, técnicas y herramientas".

Decíamos al principio de esta sub-sección que puesto que la disciplina de la ingeniería del método está comenzando a fraguarse en estos últimos años, sus problemas abarcan tanto su finalidad y contenido como la terminología que debe utilizarse durante su desarrollo. Así como en los párrafos precedentes hemos tratado los aspectos terminológicos, vamos a abordar en este momento algunas de los problemas de investigación que trata de cubrir la ingeniería del método. Para ello, vamos a referir una vez más a [Bri96], en el que, a modo de ejemplo, se presentan cuatro de los tópicos tratados dentro de la ingeniería del método: las *técnicas de metamodelización*, la *interoperabilidad de herramientas*, la *comparación de métodos y herramientas* y el estudio de *métodos situacionales*. A continuación detallaremos un poco más cada uno de ellos.

A lo largo de la presente sección estamos tratando de forma general el tema de la metamodelización. En particular, ya hemos abordado la existencia de *lenguajes* de metamodelización. Para la ingeniería del método, resulta más apropiado hablar de *técnicas* de metamodelización. Según la definición de técnica que hemos mostrado con anterioridad (definición revisada de la incluida en [Bri96]), una técnica de metamodelización sería un procedimiento -posiblemente con un lenguaje recomendado- para llevar a cabo una actividad de metamodelización. Sin embargo, y puesto que

en efecto en el momento actual esta terminología no se encuentra estandarizada, es habitual identificar el término *técnica de metamodelización* con el término *lenguaje de metamodelización*.

El segundo de los tópicos es la interoperabilidad de herramientas, relacionada intrínsecamente con la interoperabilidad de los métodos y técnicas utilizadas en dichas herramientas (véanse por ejemplo [DZ00, GV95, NBY96, Urb91]). Puesto que en efecto existen multitud de herramientas informáticas distintas que respaldan partes del proceso de desarrollo de sistemas de información, aparecen de manera natural problemas relacionados con la integración de dichas herramientas. La integración persigue objetivos tales como la comunicación entre herramientas, de tal forma que se pueda compartir información entre ellas, o incluso la integración de sistemas entre unas y otras.

Relacionado con el tema anterior está la comparación de métodos y herramientas. A la vista de la gran cantidad de métodos y herramientas disponibles en la literatura y en el mercado, ¿cómo se puede decidir cuál es el más adecuado para el desarrollo de un sistema de información concreto? La evaluación y comparación cualitativa y cuantitativa de los distintos medios puede constituir un primer paso para establecer criterios de calidad entre unos y otros, tales como completitud, expresividad, comprensibilidad, comunicabilidad, eficiencia, etc. Todos los indicios apuntan a que el marco conceptual y estructural que aporta la metamodelización puede jugar un papel fundamental para realizar las tareas de evaluación y comparación de métodos y herramientas.

El último de los tópicos de la ingeniería del método que vamos a tratar es el de una sub-área de la disciplina principal, sub-área denominada *ingeniería situacional del método* [13] . El desarrollo de un sistema de información concreto es una tarea compleja que siempre se lleva a cabo utilizando un método de desarrollo específico. Sin embargo, es poco frecuente encontrar desarrollos de sistemas que hayan utilizado, de forma estricta, un único método seleccionado entre los existentes en el mercado y la literatura. Por el contrario, el desarrollo se suele llevar a cabo utilizando, ya sea es-

[13]El término 'situacional' no existe en castellano. En este trabajo lo hemos adoptado como traducción literal del inglés 'situational' en 'situational method engineering'. En inglés, 'situational' es el adjetivo derivado del nombre 'situation'. Por tanto la traducción más correcta de 'situational' sería 'relativo a la situación', y un posible término en castellano que capturase esta idea sería 'contextual'. Sin embargo creemos que este término no recoge de manera exacta el significado del inglés 'situational', por lo que hemos creído más adecuado acuñar el término 'situacional'.

tructurada o heurísticamente, ideas y elementos de diversos métodos, de tal manera que el resultado es un *nuevo* método que se ajusta de forma precisa al sistema particular que se está desarrollando. Dentro de este contexto se enmarca la ingeniería situacional del método, que, según se indica en [tHV97], "puede ser definida como la adaptación de un método a las necesidades de un proyecto particular" o según [BSH99], "es la disciplina para construir métodos específicos de proyecto, denominados *métodos situacionales*, a partir de partes de métodos existentes, denominados *fragmentos de métodos*. Esta técnica recibe el nombre de *ensamblaje de métodos*.". Dicho de otro modo, la ingeniería situacional del método pretende establecer un marco sólido y riguroso que permita, durante el desarrollo de un sistema de información concreto, la elección, combinación, adaptación y construcción del método que sea más apropiado para ese desarrollo en particular, y de tal manera que la experiencia adquirida en ese proceso pueda ser reutilizada en proyectos sucesivos. La sub-área del estudio de métodos situacionales es uno de los tópicos de la ingeniería del método sobre el que podemos encontrar más referencias en la literatura (véanse, además de los ya citados [tHV97] y [BSH99], las referencias [HBO94] y [Har97]).

1.6. Perspectiva general de la memoria

Llegados a este punto ya contamos con un marco suficientemente extenso como para situar tanto el contenido como las aportaciones del resto de la presente memoria.

Con el objetivo de proporcionar una formulación breve, pero que va a ser obligatoriamente imprecisa, podríamos afirmar que el objetivo de esta memoria es el de *estudiar la metamodelización de ciertos aspectos de comportamiento* en relación con el desarrollo de sistemas de información. Vamos a aclarar y detallar esta afirmación. Una de las tareas más fundamentales (si no la más fundamental) que es necesario llevar a cabo para desarrollar un sistema de información es la modelización del dominio de aplicación. Por ejemplo, en [BRJ99] se dice que la modelización "es una parte central de todas las actividades que conducen a la producción de buen software".

Esta tarea de modelización puede estar orientada en dos sentidos: *modelización estructural*, centrada en la organización del sistema modelizado; o *modelización de comportamiento*, centrada en la dinámica del sistema modelizado. Esta distinción ya ha sido incluida anteriormente en este capítulo. En concreto, en la sub-sección 'Algunos tipos de lenguajes en Informática' hemos discutido brevemente este mismo

tema, mostrando que en la literatura podemos encontrar lenguajes de modelización centrados o bien en el aspecto estructural o bien en el aspecto de comportamiento. Profundizando un poco más en este asunto, observamos que en realidad ambos aspectos de la modelización no son excluyentes, sino más bien al contrario: la modelización completa de un sistema complejo necesariamente conllevará el uso tanto de la modelización estructural como de la de comportamiento. Es cierto sin embargo, que el peso de uno de los tipos de modelización puede ser mayor en función de las características del sistema (o de parte del sistema) que se esté considerando. Por ejemplo, la modelización de un sistema de bases de datos es esencialmente de tipo estructural, con algunos aspectos de comportamiento. Por el contrario, la modelización de un sistema reactivo (es decir, aquel que "está caracterizado por ser, en gran medida, dirigido-por-eventos, teniendo que reaccionar continuamente a estímulos externos e internos. Ejemplos [de este tipo de sistemas] incluyen teléfonos, automóviles, redes de comunicaciones, sistemas operativos de ordenadores, sistemas de aviónica y misiles, y el interfaz de usuario de muchos tipos de software común" [Har87]) es fundamentalmente una modelización de comportamiento, con una componente estructural menor. En cualquier caso, es innegable la existencia (la coexistencia) de ambos tipos de modelización a lo largo del desarrollo de un sistema de información. La reciente aparición y creciente aceptación del Lenguaje Unificado de Modelización (UML, Unified Modeling Language) [OMG01b, BRJ99, RJB99, JBR99] como lenguaje estándar *de facto* para el diseño y análisis orientado a objeto no es sino una prueba de ello. En efecto, en su intención de crear un lenguaje con un amplio ámbito de aplicación los creadores de UML (por una parte los conocidos como "Tres amigos" –siendo utilizada literalmente esta expresión en castellano incluso en el mundo anglosajón–, Grady Booch, Ivar Jacobson y James Rumbaugh; y por otra el Grupo de Gestión de Objetos –OMG, Object Management Group–, un "consorcio de afiliación abierta, sin ánimo de lucro, que produce y mantiene especificaciones informáticas industriales destinadas a aplicaciones de empresa interoperables"[14]) se han visto obligados a confeccionar un lenguaje complejo, que incluye sub-lenguajes destinados a la modelización estructural, por una parte, y otros sub-lenguajes destinados a la modelización de comportamiento. Este hecho ha provocado que UML sea un lenguaje con un elevado nivel de complejidad (lo que ha sido analizado por ejemplo en [SC01]) cuyo estudio ha necesitado la convocatoria de una serie de congresos

[14] http://www.omg.org

internacionales dedicados íntegramente al lenguaje [BM98, FR99, EKS00, GK01].

Por otra parte, como acabamos de ver, la ingeniería situacional del método es una disciplina que trata sobre la construcción y adaptación de métodos específicos que se ajusten a las necesidades concretas de cada proyecto, dentro de un desarrollo general de sistemas de información. Para desarrollar las tareas propias de la ingeniería situacional del método es necesario disponer de artefactos apropiados, como las técnicas de metamodelización. Una de estas tareas es la realización de modelos de alto nivel (metamodelos) de métodos, técnicas y lenguajes existentes, de forma que con dichos metamodelos se puedan realizar comparaciones, transformaciones, adaptaciones, etc. Los lenguajes que pueden ser objeto de interés para la ingeniería del método pueden ser tanto lenguajes para la modelización de estructura (a los que según hemos observado anteriormente llamaremos, por brevedad, lenguajes de estructura) como lenguajes para la modelización de comportamiento (o lenguajes de comportamiento), lo que conecta con la discusión anterior. Sin embargo, las técnicas de metamodelización existentes en la literatura han concentrado sus esfuerzos preferentemente en el desarrollo de metamodelos de lenguajes de estructura, y en algunos casos de lenguajes de comportamiento (distintos ejemplos de técnicas de metamodelización y de metamodelos tanto de lenguajes de estructura como de lenguajes de comportamiento pueden encontrarse en [MBJK90, Sae95, VtH95, KLR96, HSB97, BSH99, EHHS00, OMG01b]). En los ejemplos de metamodelos de lenguajes de comportamiento que aparecen en la literatura, éstos se centran en los aspectos de tipo estático de dichos lenguajes, sin representar en general aquellas características que a nuestro juicio distinguen esencialmente a los lenguajes de comportamiento. Estas características permiten que los modelos construidos con estos lenguajes (y que son, por tanto, modelos que representan el comportamiento del sistema modelizado) dispongan de una cierta noción de dinámica, de ejecución de tales modelos. Esta distinción, que creemos es fundamental para comprender plenamente el significado y el uso de los lenguajes de comportamiento, aparece sólo de forma implícita en la literatura. La necesidad de la expresión *explícita* de esta diferencia (así como el desarrollo de un marco que facilita su representación) es una de las aportaciones fundamentales del presente trabajo.

Tras este preámbulo el objetivo general que exponíamos al principio de la sección es más fácilmente comprensible. La presente memoria trata de estudiar la metamodelización de aspectos de comportamiento, proporcionando una solución para

el desarrollo de metamodelos de lenguajes de comportamiento que capturen todas las características de estos lenguajes, tanto las de tipo estático, como las que son esencialmente dinámicas. De forma más detallada, esta solución se ha obtenido fundamentalmente a través de dos vías. Por una parte se utiliza una técnica de metamodelización particular, la técnica de metamodelización NÓESIS [DZR97, DZ]. Esta técnica es un proyecto que está siendo desarrollado por el Grupo de Investigación NÓESIS de la Universidad de Zaragoza a lo largo de los últimos años. La técnica NÓESIS, fundamentada en una base conceptual de tipo similar a la utilizada por Sowa [Sow84], es una técnica de metamodelización que pretende ser una técnica de especial utilidad en el ámbito de la ingeniería situacional del método. Por otra parte, en esta memoria presentamos una arquitectura que trata de aportar una solución general para la representación de las características de comportamiento, especialmente para la creación de metamodelos de lenguajes para la modelización de comportamiento. Esta arquitectura puede utilizarse en diferentes niveles de abstracción, y proporciona un marco de trabajo genérico, no ligado *a priori* con ningún lenguaje de modelización ni de metamodelización, lo que mostraremos explícitamente en esta memoria. Por una parte utilizaremos la arquitectura en conjunción con la técnica de metamodelización NÓESIS, proporcionando una nueva noción de metamodelo NÓESIS que se adapta de forma adecuada a la representación de lenguajes de comportamiento. Por otra parte, utilizaremos la arquitectura en conjunción con UML como técnica de metamodelización para presentar un ejemplo de metamodelo de comportamiento expresado en UML.

En concreto, la organización de la memoria es la siguiente. En el Capítulo 2 presentamos la técnica de metamodelización NÓESIS. Es importante aclarar que esta técnica, en sí, no es una aportación de esta memoria. Si bien el autor de la memoria ha colaborado en algunos aspectos de su desarrollo, de manera global la técnica NÓESIS no es una contribución del presente trabajo. Sin embargo, para facilitar la comprensión de las ideas fundamentales de la técnica hemos optado por presentarla a través de un ejemplo concreto. Este ejemplo, que ha sido desarrollado con profundidad, sí que es una aportación de esta memoria, en dos sentidos diferentes. Por una parte es un ejemplo detallado de aplicación de la técnica NÓESIS, de tal manera que constituye una demostración práctica de sus ventajas. Por otra intenta mostrar el interés de un modelo de bases de datos, que, aunque ni ha tenido una gran aceptación práctica ni ha sido especialmente discutida en la literatura, entendemos que en

su concepción contenía diferentes ideas innovadoras que han sido después utilizadas ampliamente en el ámbito de la modelización. Este modelo de bases de datos es el *modelo RM/T*, presentado originalmente por E.F. Codd [Cod79] como una extensión del modelo relacional [Cod70], y propuesto años más tarde como futuro estándar de modelo de bases de datos, al que hipotéticamente se llegaría tras una evolución del mencionado modelo relacional [Cod90].

En el Capítulo 3 presentamos la primera aportación fundamental de esta memoria: una arquitectura para la representación de comportamiento. Esta arquitectura expresa de forma explícita la distinción entre la parte estática y la parte puramente dinámica de la representación de comportamiento. Es importante incidir en que, cuando hablamos de parte estática no nos estamos refiriendo a la modelización estructural, sino que, dentro de la modelización de comportamiento, distinguimos dos componentes, una estática y una dinámica. En el segundo capítulo se explicará con detalle esta diferenciación a través de distintos ejemplos, y se mostrará como esta distinción (y por tanto la arquitectura) es aplicable en los diferentes niveles de abstracción que se han explicado en este mismo capítulo, si bien como es lógico, la interpretación de la arquitectura será diferente en función del nivel de abstracción que se esté considerando. En particular, en el caso del Nivel de Meta-metamodelo o Nivel Axiomático (según utilicemos la terminología de la Arquitectura de Cuatro Niveles o la del Cubo de Hofstede y Verhoef), el uso de la arquitectura sobre las técnicas de metamodelización conlleva en general una modificación de la noción de metamodelo que cada técnica proponga. Más en concreto, en esta memoria mostramos cómo se puede aplicar la arquitectura para ampliar la noción de metamodelo de la técnica de metamodelización NÓESIS, de tal manera que los metamodelos NÓESIS capturen con mayor precisión las características de comportamiento de los lenguajes (meta)modelizados.

El Capítulo 4 está dedicado a la presentación detallada de un metamodelo NÓESIS del formalismo de comportamiento Statecharts. El interés de este formalismo es innegable a la vista de la amplia aceptación del mismo como formalismo para la representación y modelización de sistemas reactivos ([Har87, HPSS87, HLN+90, HRdR92, vdB94, HKCK95, HN96, MSPT96, EGKP97, HP98, RAC99, Gei99, LdBC00, CABJ01, GHD01, Sch01, FH, BGGK]), y su adaptación para la representación de comportamiento dentro de lenguajes de modelización orientados a objeto ([RBP+91, SGW94, HG97, LMM99, LP99b, LP99a, Dou99, GLM99,

Kwo00, EHHS00, BCR00]). Este metamodelo (que fue parcialmente presentado en el ECOOP'2000–International Workshop on Model Engineering [DRZ00a]) es la segunda gran aportación de esta memoria, puesto que es el primer metamodelo completo de Statecharts que existe en la actualidad. La referencia [HP98] incluye una descripción exhaustiva de la versión de Statecharts incluida en la herramienta STATEMATE, pero dicha descripción no está abordada desde una perspectiva de metamodelización (o ingeniería del método) y por tanto se encuentra desprovista de las ventajas que esta perspectiva proporciona (puesto que, por ejemplo, la metamodelización facilita la comprensibilidad y usabilidad, y faculta para la comparación y adaptación). Además, esta versión de Statecharts carece del enfoque que proporciona la arquitectura que presentamos en esta memoria, y por tanto no establece de manera clara una distinción entre la parte estática y la parte dinámica del lenguaje. Como ya hemos observado anteriormente, entendemos que la incorporación de esta distinción es fundamental para una mejor comprensión de los lenguajes de comportamiento en general, y del lenguaje Statecharts en particular.

Por su parte, la adaptación de Statecharts que bajo el nombre de *State Machines* ha sido incluida en UML [OMG01b] sí que se presenta bajo una perspectiva de metamodelización, puesto que forma parte del *metamodelo de UML*. Sin embargo esta versión también carece del enfoque que proporciona la arquitectura, lo que influye en el hecho, destacado por distintos autores [LMM99, LP99a, EHHS00], de que la semántica dinámica, o semántica de ejecución, de UML State Machines no esté definida de manera precisa. El quinto y último capítulo de esta memoria está dedicado a la presentación de un metamodelo de UML State Machines que ha sido elaborado teniendo en cuenta la arquitectura NÓESIS. La particularidad más destacada de este metamodelo es que está expresado en UML (siguiendo el estilo de definición de este lenguaje), lo que es una prueba de la flexibilidad de la arquitectura NÓESIS para ser utilizada con independencia de los lenguajes de modelización y metamodelización considerados. Las ideas fundamentales de este metamodelo (si bien utilizando la técnica NÓESIS como técnica de metamodelización) fueron presentadas en el UML'2000–Workshop on Dynamic Behaviour in UML Models: Semantic Questions [DRZ00b] y han sido recientemente aceptadas, tal y como aparecen en esta memoria, para su publicación en la revista *Journal of Database Management* [DRZ].

Capítulo 2

Un metamodelo Nóesis de RM/T

Una de las partes fundamentales (si no la más fundamental) en la compleja tarea que es el desarrollo de un sistema de información es la *modelización* del dominio de aplicación, para lo que es especialmente interesante disponer del método (y/o técnica y/o lenguaje) más adecuado para cada proyecto concreto. Esta elección está condicionada por multitud de factores, tales como el tipo de dominio de aplicación, la fase de desarrollo en la que se encuentre el proyecto, los medios humanos y físicos disponibles, etc. Debido a estas dificultades, en los últimos años están surgiendo diferentes disciplinas que tratan de realizar avances en este sentido. En particular, la *ingeniería del método* [Bri96, BLW96] es la disciplina que trata sobre todos los aspectos relacionados con el diseño, creación y adaptación de métodos, técnicas y herramientas para el desarrollo de sistemas de información. Uno de estos aspectos consiste en el desarrollo de técnicas específicas que permitan la expresión y representación de los métodos y lenguajes citados con anterioridad. Así como a la realización de modelos *de primer nivel* (es decir, modelos del dominio de aplicación) se la denomina modelización, la realización de modelos *de segundo nivel* (es decir, de los métodos, técnicas y lenguajes utilizados para crear los modelos de primer nivel) se denomina *metamodelización*. Por tanto, el desarrollo de técnicas de metamodelización adecuadas es una de las tareas de la ingeniería del método.

En este capítulo presentamos las características principales de la técnica de metamodelización Nóesis [DZR97, DZ], una técnica que está especialmente adaptada a las necesidades de la ingeniería del método, y más en particular a la sub-disciplina de la ingeniería situacional del método [tHV97, BSH99], la cual trata a su vez de la adaptación de métodos para las necesidades específicas de proyectos de desarrollo

concretos. En este trabajo vamos a mostrar los aspectos principales de la técnica NÓESIS fundamentalmente a través de un ejemplo, en concreto mostrando un metamodelo NÓESIS del modelo de bases de datos RM/T [Cod79]. La elección de este ejemplo ha tenido una motivación múltiple. Por una parte hemos buscado un modelo[1] del que no existe ningún metamodelo en la literatura, de tal forma que el metamodelo que presentamos supone una aportación novedosa con respecto a lo escrito hasta el momento sobre RM/T (hemos de observar, sin embargo, que obviamente este planteamiento tiene el inconveniente de impedir la comparación con otros metamodelos como método de evaluación de la calidad de lo presentado). Otra razón para la elección de RM/T, relacionada parcialmente con la anterior, es la defensa de ciertas ideas que aporta RM/T que entendemos pueden tener validez para el desarrollo de bases de datos. Como veremos más adelante, el modelo RM/T ha tenido una muy reducida aplicación práctica conocida, y ha sido escasamente referido y tratado en la literatura. Sin embargo, algunas de las ideas que recoge RM/T han sido más tarde profusamente utilizadas en otros ámbitos de la modelización y las Ciencias de la Computación, aunque sin hacer referencia al modelo creado por E.F. Codd. Una última razón para haber elegido RM/T es que, a pesar de no ser un modelo especialmente complicado, tiene un nivel de complejidad suficiente como para poder mostrar la mayoría de las características esenciales de la técnica NÓESIS.

2.1. Introducción a la técnica de metamodelización Nóesis

La técnica NÓESIS es una técnica de metamodelización desarrollada por el grupo de investigación NÓESIS de la Universidad de Zaragoza. Una primera versión de la técnica fue presentada en el Noveno Congreso sobre Ingeniería Avanzada de Sistemas de Información (CAiSE), celebrado en 1997 en Barcelona [DZR97]. En los últimos años la técnica ha sido mejorada con nuevas funcionalidades [DZ], lo que recien-

[1]Recordemos que según se ha visto en el Capítulo 1, referirse a un modelo de bases de datos con los términos 'modelo' o 'lenguaje' (o incluso 'técnica') es dependiente del punto de vista bajo el que se esté abordando el problema. Aunque en principio nada impediría –y de hecho sería probablemente más coherente con las discusiones realizadas anteriormente– referirnos al 'lenguaje de bases de datos RM/T', mantendremos preferentemente la expresión 'modelo RM/T' para no confundir al lector con respecto a las referencias más habituales en la literatura

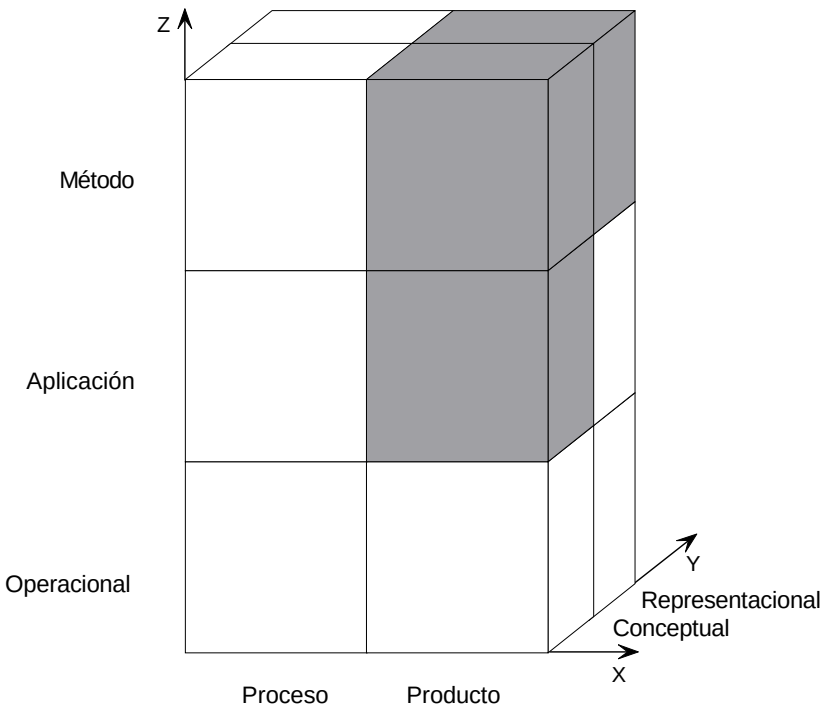


Figura 2.1: Dimensiones de modelización tratadas en la técnica NÓESIS

temente ha permitido mostrar sus ventajas en diversos encuentros internacionales [DRZ00a, DRZ00b, DZE00, DZ00]. Con la inclusión de estas nuevas características, la técnica NÓESIS se ha convertido en un instrumento de metamodelización especialmente adaptado para las tareas de la ingeniería situacional del método.

El desarrollo de la técnica NÓESIS ha estado guiado por un espíritu principal: mejorar la legibilidad, comprensibilidad, usabilidad y expresividad de los metamodelos. En general, el enfoque que aporta la metamodelización ha sido reconocido como un enfoque válido para la mejora de esas características respecto a la representación del conocimiento sobre métodos [Ver93, tHV97]. La diferencia de base que introduce la técnica NÓESIS respecto a otras técnicas de metamodelización es que utiliza como primitiva epistemológica la noción de *concepto*, ya que entendemos que, puesto que ésta es una de las nociones más próximas al conocimiento del ser humano, su uso puede mejorar aún más las características a las que hacíamos referencia anteriormente con respecto a los metamodelos.

Antes de comenzar con el detalle de algunos de los elementos que caracterizan a la técnica NÓESIS, es necesario realizar varios comentarios que ponen en relación

nuestra técnica con determinados aspectos tratados en los Preliminares. En primer lugar, hemos observado en el Capítulo 1 que es más habitual utilizar el término *técnica* que el término *lenguaje* al hablar de artefactos de metamodelización. Sin embargo, si deseáramos seguir de forma estricta la terminología de la ingeniería del método propuesta por Brinkkemper, en nuestro caso no sería totalmente correcto hablar de *técnica* NÓESIS. Esto es así puesto que recordemos que, siempre según la definición de Brinkkemper, una técnica es un *procedimiento*, con una *notación (lenguaje)* recomendada, para llevar a cabo una actividad de desarrollo. En nuestro caso la 'actividad de desarrollo' es una 'actividad de metamodelización', y por tanto una técnica de metamodelización debería proporcionar un procedimiento concreto para realizar la actividad de metamodelización. Como mostraremos a lo largo de las próximas secciones, con la presentación de un ejemplo de metamodelo NÓESIS, lo que proporcionamos es esencialmente un lenguaje para la elaboración de metamodelos, pero no un procedimiento que sirva de guía a cómo han de ser creados los metamodelos. Otra forma de interpretar esta situación la proporciona el cubo de Hofstede y Verhoef que también hemos analizado en el Capítulo 1 (la figura 2.1 es una ligera modificación de la figura 1.4 -página 26- del cubo original; en esta misma sección se explica más adelante el significado de la figura 2.1). Recordemos que una de las dimensiones de modelización que se incluyen en este enfoque distingue entre los aspectos 'producto' y 'proceso'. Siguiendo de forma rigurosa el enfoque de Hofstede y Verhoef, cualquier técnica de metamodelización, puesto que pertenece al Nivel Axiomático (como ya hemos comentado en los Preliminares, este nivel –que no aparece en el cubo de la figura 1.4 y por tanto tampoco en la figura 2.1– se sitúa por encima del Nivel del Método), debería cubrir tanto el aspecto producto como el aspecto proceso del Nivel del Método. Es decir, debería proporcionar una definición de metamodelo (el qué, el producto) y también una forma de construcción de metamodelos (el cómo, el proceso). En el caso particular que nos ocupa, la técnica NÓESIS proporciona por una parte una definición detallada de metamodelo, es decir, define explícitamente el producto de la metamodelización. Por otra parte, el proceso de la metamodelización es recogido en la última versión de la técnica NÓESIS [DZ] mediante la adjunción de una familia de transformaciones que permite la manipulación (es decir, la creación y modificación) de metamodelos NÓESIS. Sin embargo, el enfoque NÓESIS no proporciona en el momento actual ninguna guía metodológica concreta que un meta–analista deba seguir a la hora de utilizar la técnica NÓESIS.

Como conclusión de esta primera observación, diremos que, aunque admitimos que NÓESIS es probablemente más un lenguaje que una técnica, mantendremos el término *técnica* NÓESIS por razones históricas y como forma de mantener la coherencia con respecto a la terminología que se encuentra con mayor frecuencia en la literatura.

El segundo aspecto que es necesario tratar con respecto a lo visto en el Capítulo 1 es el relativo a la discusión sobre la definición de los lenguajes (de modelización y por tanto de metamodelización, en particular). Según hemos visto, para definir un lenguaje de forma completa es necesario proporcionar tanto su sintaxis -abstracta- como su semántica. Además, y aunque en principio no es un requisito imprescindible, todos los lenguajes proporcionan una sintaxis concreta, una notación ya sea textual o gráfica, la cual es utilizada para presentar de manera simultánea la sintaxis abstracta. En las secciones siguientes usaremos este mismo enfoque para mostrar los elementos sintácticos básicos de la técnica NÓESIS, presentando dichos elementos junto con una notación concreta para los mismos. En el caso particular de la técnica NÓESIS, como veremos de inmediato, el tipo de notación utilizada es en parte textual y en parte gráfica. Con respecto a la semántica, es importante observar que en este trabajo vamos a presentar una semántica orientada al usuario humano. Como también se ha comentado en los Preliminares al analizar cualquier aspecto relacionado con la Informática es conveniente no olvidar que esta ciencia trata con dos actores principales muy diferentes: el ser humano y el ordenador. En el caso de los lenguajes de modelización, que además de ser utilizados por usuarios humanos pueden ser manipulados por ordenadores (mediante el desarrollo de herramientas CASE, por ejemplo), un enfoque que entendemos fundamental para conseguir un correcto uso de dichos lenguajes es el de proporcionar dos semánticas para cada lenguaje, una orientada al usuario y otra orientada a la máquina. Obviamente ambas perspectivas deben coexistir en un mismo entorno, y por tanto debe lograrse una concordancia entre ellas con el objetivo principal de garantizar la comunicación hombre-máquina. Volviendo al caso concreto de la técnica NÓESIS, insistimos en que el tipo de semántica que vamos a proporcionar en esta memoria está orientada al usuario humano. Esto es así puesto que como ya hemos observado la noción fundamental que hemos decidido usar es la noción de *concepto*, y además en nuestra presentación utilizaremos fundamentalmente el lenguaje natural. La semántica orientada al ordenador de la técnica NÓESIS se encuentra en la actualidad en proceso de desarrollo.

Esta semántica está siendo definida de modo más formal, utilizando como primitiva la noción de *símbolo*, por entender que esta noción es la más cercana a la máquina.

El último aspecto que vamos a comentar es el ámbito de aplicación de la técnica NÓESIS, que como vamos a ver de inmediato se encuentra parcialmente relacionado con los dos comentarios que acabamos de realizar. En la figura 2.1 se muestra, dentro del cubo de Hofstede y Verhoef, el ámbito de aplicación de la técnica NÓESIS. El hecho de que la técnica NÓESIS se ocupa principalmente del aspecto producto del Nivel del Método ya ha sido observado con anterioridad en esta misma sección. Además, y puesto que vamos a proporcionar una sintaxis concreta, una notación, para los metamodelos NÓESIS, también aparece sombreada la dimensión correspondiente a los aspectos de representación de productos en el Nivel del Método. Por último, es necesario remarcar que al pasar al Nivel de Aplicación, la técnica NÓESIS sigue concentrándose en el aspecto producto. Observemos que el aspecto proceso se refiere, en el Nivel de Aplicación, a la forma cómo se crean los modelos, que son por su parte los productos de este nivel. Es decir, al hablar del 'proceso' en el Nivel de Aplicación estamos hablando del *proceso de modelización tradicional*, de las tareas que es necesario realizar para crear un modelo. Pues bien, la técnica NÓESIS permite desarrollar metamodelos, que a su vez referirán a una familia de modelos, pero la técnica, al menos en el momento actual, *no está ideada para que dichos metamodelos capturen información sobre el* proceso *de creación de los modelos a los que cada metamodelo refiere*. Por ello, en el Nivel de Aplicación hemos sombreado exclusivamente el sector correspondiente al aspecto producto.

2.1.1. Características principales de la técnica Nóesis

Una vez hechas las puntualizaciones de la sección anterior, vamos a presentar de forma breve las principales características de la técnica NÓESIS, que serán desarrolladas con detalle durante la presentación del ejemplo. La fundamentación de la técnica NÓESIS utiliza un enfoque similar al de Sowa [Sow84], tomando la noción de *concepto* como primitiva epistemológica. Al igual que Sowa, entendemos que cada concepto es una representación mental de una categoría de objetos[2] en el dominio de aplicación (o universo de discurso). Es importante observar que en escasas oca-

[2]Es conveniente aclarar, para evitar cualquier malentendido, que el término *objeto* es utilizado aquí en el sentido amplio de *elemento, cosa*, sin guardar relación alguna con el significado del término *objeto* del ámbito de la *orientación a objetos*.

siones un concepto determinado tiene un significado universal, sino que un mismo concepto puede referirse a unos objetos u otros en función de cual sea el dominio de aplicación. Otra característica interesante de nuestra perspectiva es que entendemos que es importante diferenciar, de forma explícita, entre las cosas referidas por el concepto, a las que denominamos *individuos del concepto*, y las representaciones de dichas cosas, a las que denominamos *instancias*. Nuestro enfoque se diferencia en otros aspectos con el de Sowa, puesto que él describe cada concepto mediante una etiqueta de tipo y un referente, mientras que la técnica NÓESIS establece que un concepto se representa por medio de un nombre y una descripción. Un *nombre* es un objeto lingüístico (es decir, una expresión gramaticalmente correcta en lenguaje natural escrito) que refiere a aquello que es representado por el concepto. Una *descripción* es un objeto lingüístico o una estructura lingüística que se usa con la intención de describir aquello que el concepto representa. En general cualquiera de los dos elementos constituyentes de la representación de un concepto puede omitirse para un concepto concreto. Esto da lugar a una pequeña categorización de tipos de conceptos, distinguiendo entre *conceptos no-descritos* (aquellos para los que sólo se proporciona nombre), *conceptos no-nominados* (aquellos para los que sólo se proporciona descripción; son muy poco frecuentes, en general) y *conceptos plenos* (o simplemente *conceptos*, aquellos para los que se proporciona tanto nombre como descripción).

Tomando como base este enfoque conceptual, un metamodelo construido con la técnica NÓESIS, es decir, un *metamodelo* NÓESIS, está constituido por cuatro componentes:

- perspectiva
- sistema de referencia
- marco representacional[3], y
- definición de modelo,

cuya utilidad y significado vamos a detallar a continuación.

[3]De forma similar a como ocurre con el término 'situacional', en castellano no existe el término 'representacional', traducido de forma literal del original en inglés 'representational'. En este caso sí existe una traducción exacta del término, que es 'figurativo'. Sin embargo, en este caso creemos que el término 'marco figurativo' no expresa con suficiente claridad la idea que se encuentra detrás de esta componente, y de ahí que hayamos creído conveniente adoptar el término 'representacional'.

Perspectiva

La *perspectiva* de un metamodelo NÓESIS es una breve descripción, en lenguaje natural, del contenido y propósito del metamodelo. En la perspectiva se describe de manera resumida el lenguaje metamodelizado, con la intención fundamental de centrar el contexto en el que el metamodelo ha sido desarrollado. De hecho, en la presentación de la práctica totalidad de los metamodelos que pueden encontrarse en la literatura se incluye de una forma u otra esta característica, pues es claro que es necesario informar al meta–analista del contexto en el que sitúa el metamodelo en cuestión. La principal aportación que hace la técnica NÓESIS en este sentido es el de la *explicitación* de este artefacto, es decir, la obligatoriedad de ser incluida como componente de cualquier metamodelo NÓESIS. Un ejemplo de perspectiva se muestra en la página 73 para el metamodelo NÓESIS de RM/T.

Sistema de Referencia

La segunda componente de un metamodelo NÓESIS es el sistema de referencia. El *sistema de referencia* es una relación detallada de todos los conceptos del lenguaje analizado que son utilizados en el metamodelo. Aunque en principio *todos* los conceptos deberían ser incluidos, en la práctica, y con el objetivo de mejorar la usabilidad y comprensibilidad del metamodelo (que recordemos es el leitmotiv del desarrollo de la técnica NÓESIS), se incluyen en el sistema de referencia sólo aquellos conceptos que el meta–analista entienda que puedan dar lugar a ambigüedades. En otras palabras, el sistema de referencia se utiliza para incorporar conocimiento terminológico del dominio (lenguaje) que el metamodelo captura, mejorando su expresividad y reduciendo el riesgo de interpretaciones erróneas. En este sentido, el sistema de referencia desempeña un papel similar, aunque en un nivel de abstracción superior, al de las 'bases de conceptos' presentadas en [BF94].

Es necesario hacer notar que en la primera versión publicada de la técnica NÓESIS [DZR97] el sistema de referencia recibía la denominación de *sistema de conceptos*. El término sistema de referencia se introduce como traducción libre del inglés 'anchoring system', cuya traducción literal sería 'sistema de sujeción' o 'sistema de anclaje'. Este término inglés ha sido tomado de las referencias [BSH99, Har97], en las que un 'anchoring system' es definido como un conjunto restringido de primitivas atómicas bien definidas, que se presenta con la intención de ser usado para describir

la semántica de un *conjunto limitado* de fragmentos de métodos (recordemos que la noción de *fragmento de método -method fragment-* es una de las nociones claves de la ingeniería situacional del método). En nuestro caso, el sistema de referencia se utiliza para referir a una familia de conceptos bien definidos relacionados con un metamodelo *concreto*. El término 'anchoring system' se relaciona habitualmente también con enfoques ontológicos, lo que en el caso de la ingeniería del método es tratado por ejemplo en [BSH99] y [Wan96].

Concepto pleno	<nombre>*<descripción>*
Concepto no-descrito	<nombre>**
Concepto no-nominado	*<descripción>*

Tabla 2.1: Notación NÓESIS para la representación textual de conceptos

La sintaxis concreta (notación) que se utiliza para la expresión de los sistemas de referencia es de nuevo esencialmente textual, aunque es necesario hacer algunas precisiones sobre esta representación. En primer lugar, un sistema de referencia se presenta mediante una tabla de dos columnas, con tantas filas como conceptos se incluyan en el sistema de referencia (un ejemplo de sistema de referencia se muestra en la página 115 para el metamodelo NÓESIS de RM/T). En la primera columna aparecen los conceptos propiamente dichos, utilizando en concreto la notación que se muestra en el Cuadro 2.1, en función del tipo de concepto de que se trate. En todas estas notaciones, el asterisco (*) es un carácter reservado que por tanto no puede formar parte de los objetos lingüísticos que constituyen <nombre> y <descripción>. Además, en el caso de que el nombre de un concepto esté formado por más de una palabra, deben reemplazarse los espacios en blanco entre palabras por el signo de subrayado (_), que es por tanto es también en un símbolo reservado dentro de la notación utilizada para expresar los conceptos. Por su parte, en la segunda columna de un sistema de referencia se muestra, para cada concepto, la referencia completa (incluyendo además del identificador, el número de página y número de sección si lo hubiera) del documento del cual se ha extraído dicho concepto. En general, esta referencia será de cualquier tipo de documento (libro, artículo, borrador técnico, documento de especificaciones, dirección de Internet, etc.) que sea la fuente original del lenguaje metamodelizado, admitiendo la posibilidad de que para un mismo

concepto se incluyan varias referencias distintas. Esto es así puesto que es posible que un concepto inicialmente definido en una cierta referencia haya sido redefinido o clarificado en una segunda. Entendemos que la inclusión de estas referencias pueden ser de utilidad para el analista interesado en una profundización de los motivos de aparición de determinados conceptos, además de servir de control de calidad durante y tras la elaboración del metamodelo. Además de esta(s) referencia(s), en la segunda columna de cada concepto en el sistema de referencia pueden aparecer las siglas '*int.*', lo que significa que el concepto en cuestión se ha incluido bajo la *interpretación* particular del meta–analista. La inclusión de esta interpretación puede ser debida o bien a que el concepto no está descrito con total precisión en las fuentes originales o bien a que el meta–analista aprecia que es necesaria dicha interpretación particular para una mejor comprensión de ese concepto en concreto.

El segundo aspecto que es interesante revisar respecto a la forma en la que se construye un sistema de referencia es la distinción que se realiza entre tipos de conceptos en función del tipo de conocimiento que recogen. De partida se realiza una razonable asunción previa, y es que el analista (usuario del sistema de referencia) dispone de un mínimo conocimiento epistemológico previo. Por tanto, se admite que en la descripción de cada concepto del sistema de referencia se incluyan *conceptos epistemológicos*, es decir, conceptos no-descritos que pertenecen al conocimiento común o general. Se asume por tanto que estos conceptos de base no necesitan de ninguna explicación adicional, y no son incluidos de forma explícita en un sistema de referencia. Conceptos epistemológicos serían, por ejemplo, la noción de nombre *(de algo)* o la noción de par *(de objetos)*. Utilizando como piezas elementales estos conceptos básicos, se pueden empezar a construir los conceptos que propiamente forman parte de un sistema de referencia, que a su vez se clasifican en conceptos primitivos y conceptos derivados.

Un *concepto primitivo* es aquel que se basa exclusivamente en conocimiento epistemológico. En general, un concepto primitivo puede ser no-descrito o pleno (obviamente, y puesto que se trata de conceptos de base, los conceptos primitivos no pueden ser no-nominados). Si el concepto primitivo es no-descrito, su inclusión explícita en el sistema de referencia se deberá a que el meta–analista lo considera como un elemento esencial del metamodelo, y que, a pesar de no precisar descripción, es importante que se tenga presente como elemento constitutivo del metamodelo. Si por el contrario un concepto primitivo es pleno, es decir, incluye descripción, esta des-

cripción estará escrita en términos epistemológicos, es decir, en términos de lenguaje natural de interpretación evidente, y en ningún caso basados en otros conceptos del sistema de referencia.

Por el contrario, un *concepto derivado* es siempre un concepto descrito (puede ser por tanto no-nominado o pleno), y en su descripción pueden aparecer tanto conceptos epistemológicos como conceptos primitivos, así como, a su vez, otros conceptos derivados del sistema de referencia. Dicho de otro modo, un concepto derivado siempre 'se deduce' de otros conceptos del sistema. Este razonamiento se formaliza estableciendo una relación de derivación entre los conceptos de un sistema de referencia. Un concepto C' es *directamente derivado* de otro concepto C si la descripción de C' contiene el nombre de C. La *relación de derivación* propiamente dicha es el cierre transitivo de la relación de derivación directa. Esta propiedad, en cierto modo 'teórica', tiene una aplicación práctica directa, relativa a la forma de presentación de la lista de conceptos que conforman un sistema de referencia. Aunque en principio la técnica NÓESIS no restringe en ningún sentido el orden en que deben presentarse los conceptos, existen dos ordenaciones que aparecen de manera natural: en primer lugar el orden alfabético, el cual es útil especialmente cuando al sistema de referencia se le da la funcionalidad de un diccionario común, es decir, cuando se busca el significado de un concepto concreto dentro del sistema; y en segundo lugar el orden que determina la relación de derivación, según el cual en la lista de conceptos aparecerán primero los conceptos primitivos, después los conceptos derivados directamente de los conceptos primitivos, etc. Este segundo criterio de ordenación, a pesar de no proporcionar una relación de orden total, mejora la legibilidad del sistema de referencia y por tanto incrementa la comprensibilidad de los conceptos. Entendemos que una especificación completa de un metamodelo NÓESIS debe incluir ambas formas de acceso a los conceptos, puesto que ambas pueden ser útiles en distintas fases del análisis del metamodelo.

Otro aspecto ligado a los conceptos derivados y la relación de derivación se refiere a la posibilidad de aparición de circularidades dentro del sistema de referencia. Una *circularidad* dentro de un sistema de referencia se da cuando un concepto C es derivado (directa o indirectamente) de un concepto C' y a su vez este concepto C' es derivado del concepto C inicial. Otros aspectos relacionados íntimamente con este problema ya han sido abordados en los Preliminares, donde hemos constatado el hecho obvio de que un diccionario tradicional está construido en esencia a base de

definiciones circulares. Entendemos que en el ámbito de un sistema de referencia, en el que el número de conceptos no es en general excesivamente grande, las circularidades deben ser evitadas con el fin básico de mejorar la comprensibilidad del sistema. Esto no quiere decir sin embargo que el uso de definiciones circulares esté prohibido en un sistema de referencia. Entendemos que en determinados casos una definición circular correctamente fundada puede facilitar la comprensión de ciertos conceptos cuya descripción sería enormemente compleja en caso de no admitir la anomalía circular. Estos conceptos son fundamentalmente aquellos cuya definición es *recursiva* en su esencia (observemos que este es un caso particular de la descripción de circularidad dada anteriormente, en el que C y C' son el mismo concepto). Ejemplos de este tipo de conceptos son la noción de *expresión* habitual en muchos lenguajes de programación (la cual podría estar expresada como expresión∗número, variable u operación algebraica entre expresiones∗) o la noción de *fórmula* que recoge por ejemplo el lenguaje PSM [tHvdW93] (cuya expresión conceptual podría ser fórmula∗variable simple o función de fórmulas más sencillas∗).

Para concluir con las observaciones relativas al sistema de referencia, es interesante destacar que el establecimiento de la relación de derivación plantea interesantes cuestiones relacionadas con la automatización y el soporte computerizado del sistema. Aunque en el momento actual del desarrollo de la técnica NÓESIS los sistemas de referencia se escriben con el apoyo elemental de programas de proceso de textos, existen varias posibilidades que pueden facilitar la creación y el análisis de los sistemas. Una de estas posibilidades consiste en la elaboración de un código de colores para identificar y diferenciar visualmente de forma rápida los conceptos primitivos de los derivados, aquellos en los que existan circularidades, etc. (un enfoque de modelización que también utiliza colores, pero aplicado a la modelización estructural de sistemas utilizando Java y UML se puede encontrar en [CLL99]). Otra posibilidad se basa en el uso de la tecnología hipertexto para facilitar la legibilidad del sistema de referencia. Por ejemplo, el sistema de referencia podría estar escrito en una tabla escrita en lenguaje HTML de tal forma que dentro de la descripción de cada concepto derivado aparecieran enlaces en los nombres de los conceptos de los cuales el concepto inicial se deriva, enlaces que lógicamente nos conducirían a la descripción del concepto original. De la misma forma las referencias que se adjuntan a cada concepto podrían ser enlaces a los documentos originales. Relacionada con este enfoque se encuentra la posibilidad de representar el sistema de referencia

mediante lenguaje XML, lo que abre la puerta a su implementación como base de datos. Más específicamente, un sistema de referencia puede entenderse como una base de conocimientos, a partir de la cual se puedan realizar inferencias automatizadas. De hecho, dentro del Grupo NÓESIS se desarrolló un prototipo (en lenguaje Lisp) en el que el grafo de derivación de sistemas de referencia de la versión inicial de la técnica NÓESIS era calculado de forma semi-automática. Mediante este grafo, el prototipo podía deducir diferentes propiedades del sistema de referencia, tales como los conceptos primitivos y los derivados, la determinación de la existencia o no de circularidades, o el cálculo de la lista de los conceptos de los que un concepto dado se deriva. Por diferentes razones este desarrollo no ha tenido continuidad en años posteriores.

Marco Representacional: Restricciones Locales

La tercera componente de un metamodelo NÓESIS, y en cierto modo la más importante, es el marco representacional. Un *marco representacional* se compone, a su vez, de un conjunto de soportes y de un conjunto de restricciones locales. La noción de *soporte* juega un papel clave en la definición de metamodelo NÓESIS, y por ello le vamos a dedicar una sección específica a continuación. Por el momento, adelantaremos que un soporte es una familia de conceptos (que serán o bien conceptos epistemológicos o bien habrán sido especificados en el sistema de referencia) que están interrelacionados a través de relaciones ISA, relaciones de atribución y referencias estructurales. El propósito de un soporte es el de establecer los elementos representacionales del lenguaje metamodelizado cuyas instancias pueden formar parte de los modelos construidos con el lenguaje. Como hemos dicho, el marco representacional contiene en general un conjunto de soportes, y junto a éstos, un conjunto de restricciones locales (un ejemplo de conjunto de restricciones locales se muestra en la página 119 para el metamodelo NÓESIS de RM/T.). Una *restricción local* es una condición, expresada en lenguaje natural escrito, que las instancias de los conceptos del soporte a las que hacíamos referencia deben cumplir con respecto a otras instancias que pertenezcan al mismo modelo. La denominación de *local* es debida a que no establecen restricciones sobre el modelo en su conjunto, sino sólo sobre una parte de él. Como veremos a continuación, una de las razones que motivan la inclusión de este tipo de restricciones es que los soportes de representan en forma de diagrama, y existen ciertas restricciones que serían muy complejas de ex-

presar en forma gráfica. En este sentido, las restricciones de un metamodelo NÓESIS (tanto las locales como las globales que estudiaremos más adelante) se asemejan en cierta medida a las restricciones que en UML se expresan utilizando el lenguaje OCL [WK99, OMG01b]. Sin embargo, y una vez más con el objetivo de mejorar la legibilidad y comprensibilidad de los metamodelos, las restricciones NÓESIS no están expresadas utilizando un lenguaje formal (este lenguaje, sin embargo, es un requisito obligado para expresar la semántica orientada a la máquina de la técnica NÓESIS, que como hemos observado está actualmente en proceso de desarrollo).

Marco Representacional: la noción de Soporte

Como acabamos de ver, un marco representacional incluye un conjunto de soportes. La noción de soporte es posiblemente la más importante entre las que aporta la técnica NÓESIS, y aunque en principio no es tan novedosa como puedan ser la perspectiva o el sistema de referencia (ya que estas dos componentes son raramente consideradas en otras técnicas de metamodelización), recoge ciertos aspectos que son poco frecuentes en la literatura. Como ya hemos observado en la sub-sección anterior, básicamente un soporte es una familia de conceptos (en general del sistema de referencia, aunque en determinados casos puede recoger también conceptos epistemológicos) interrelacionados a través de relaciones ISA, relaciones de atribución y referencias estructurales (ejemplos de soportes se muestran en la página 64 para la noción de fórmula y en la página 114 para el metamodelo NÓESIS de RM/T). Dentro de los conceptos de un soporte distinguimos dos tipos: conceptos de soporte propiamente dichos y atributos.

Un *concepto de soporte* refiere a una clase de individuos que pueden ser representados como elementos de un modelo, con independencia de los aspectos relativos a la descripción de dichos elementos. Por su parte, un *atributo* refiere a algún aspecto que describe a los individuos referidos por un concepto de soporte. En otras palabras, los conceptos de soporte refieren a *qué* puede ser representado y los atributos refieren a *los medios a través de los que* es descrito ese 'qué'. Esta distinción no supone una clasificación de separación estricta. Como se ha observado en el Capítulo 1, modelizar (y por tanto metamodelizar) son tareas con una importante carga subjetiva, y por tanto, la decisión de considerar un concepto concreto o bien como concepto de soporte o bien como atributo dependerá fuertemente de la

percepción que el meta–analista tenga del problema[4]. Esto puede llevar a que diferentes meta–analistas construyan diferentes metamodelos, y en particular soportes, para un mismo lenguaje, de la misma forma que diferentes analistas pueden crear diferentes modelos para un mismo dominio de aplicación.

Por otra parte, dentro de los conceptos de soporte, es necesario distinguir entre conceptos específicos y auxiliares, en función de si se corresponden o no, de manera directa, con elementos del lenguaje modelizado. Un concepto de soporte *específico* es el que refiere a elementos propios del lenguaje, mientras que un concepto *auxiliar* no se corresponde con ningún elemento que sea considerado explícitamente en el lenguaje, pero que es incluido en el soporte con la intención de facilitar y simplificar su construcción y posterior análisis.

Los conceptos de soporte están relacionados entre sí por medio de especializaciones que constituyen una *jerarquía ISA de conceptos*. Usando terminología propia de la teoría de grafos (véase por ejemplo [Ber91]), una jerarquía ISA es un grafo conexo dirigido acíclico con sólo un sumidero, es decir, con un único nodo que tiene grado exterior cero, y de tal forma que los arcos del grafo representan relaciones ISA (de especialización) entre los conceptos asociados a cada nodo. Además de esta propiedad, a un soporte se le exige el siguiente principio, llamado de *especialización total*: cada instancia de un concepto no–fuente de un soporte es instancia de alguna especialización de dicho concepto (donde llamamos *concepto no–fuente* a cada nodo de la jerarquía con grado interior mayor que cero, y *concepto fuente* a cada nodo con grado interior igual a cero). Ni el principio de especialización total ni otros similares son habitualmente impuestos en la literatura para las jerarquías ISA (véanse por ejemplo [BMW84], [Bra83] y [WtHvO92]), pero sin embargo entendemos que este principio mejora la comprensibilidad de la estructura y facilita su manejo. Además, este principio permite percibir una jerarquía ISA como clasificación (no necesariamente disjunta, y a diferentes niveles) de las instancias de los conceptos fuente. De

[4]Esta circunstancia no es, sin duda, un aspecto novedoso ni de la técnica Nóesis ni de las técnicas de metamodelización en general. En la esencia de incluso los más utilizados y estudiados lenguajes de modelización ya se encuentran disyuntivas similares. Sirva como muestra el tradicional ejemplo de la idea de 'matrimonio' dentro de un contexto de modelización E/R: en función de la percepción y necesidades del diseñador, un matrimonio puede ser considerado o bien una entidad o bien una asociación entre entidades (personas). De hecho, este tipo de ejemplos es utilizado por algunos autores como argumento para señalar las debilidades del modelo E/R [Dat01], aunque de igual manera puedan ser utilizados para defender su flexibilidad.

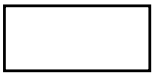
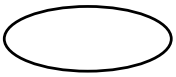
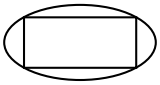
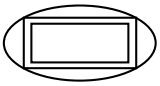
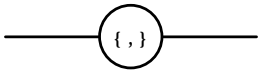
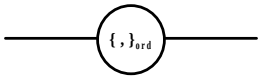
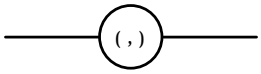
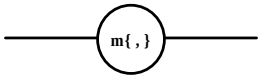
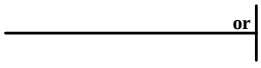
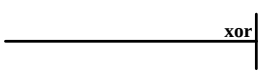
hecho, la imposición del principio es una de las principales características del enfoque que aporta la técnica NÓESIS. El principio permite que las componentes de los modelos (del Nivel de Aplicación) se representen exclusivamente por medio de instancias de los conceptos fuente, de tal forma que los conceptos no–fuente deben considerarse como organización de los conceptos fuente, organización que tiene por objetivo mejorar la usabilidad y adaptabilidad de los soportes y como consecuencia de los metamodelos.

La estructura jerárquica de los soportes en la técnica NÓESIS permite que sean fácilmente representables de manera gráfica, a través de un diagrama. En particular, los conceptos de soporte se representan de forma diferente en función de si se trata de conceptos específicos o auxiliares. En el cuadro 2.2 se muestra la notación gráfica que es utilizada en cada caso. En concreto, los conceptos específicos se representan situando su nombre en el interior de un rectángulo, mientras que los conceptos auxiliares se representan situando su nombre junto a un pequeño cuadrado.

Por otra parte, los aspectos que pertenecen a la descripción de los conceptos de soporte son capturados a través de atributos. Cada concepto de la jerarquía ISA puede ser descrito a través de otros conceptos, que en estos casos desempeñan el papel de atributos. Como vamos a mostrar a continuación, los atributos pueden ser utilizados tanto en modo simple y directo como en modos bastante complejos. Este aspecto, que es también uno de los fundamentales de la especificación de los soportes, conlleva que la técnica NÓESIS esté provista de una notable riqueza expresiva.

De acuerdo con el enfoque básico de la noción de atributo dentro de la técnica NÓESIS, consideramos que 'algo' se percibe como atributo sólo si este 'algo' es percibido como descriptor de algún otro 'algo'. Esto significa que dentro de la técnica NÓESIS se considera carente de sentido hablar de un atributo sin más, sino que un atributo siempre refiere por una parte al concepto al que describe (concepto que en este contexto es llamado *concepto atributado*) y por otra a la relación de atribución percibida. Una consecuencia práctica de este planteamiento es que en cualquier representación de un atributo, éste no puede aparecer aislado, sino que debe ir acompañado del concepto atributado y de la relación de atribución. Mostraremos un poco más adelante las posibles representaciones gráficas de atributos y relaciones de atribución. En cuanto a la representación textual, un atributo en NÓESIS se representa en la forma (A) DE C, donde C es el concepto atributado, A es el concepto que atribuye y el artículo 'de' (en lenguaje natural) representa en general la relación de

Tabla 2.2: Notación NÓESIS para la representación gráfica de soportes

NOMBRE	REPRESENTACIÓN
concepto de soporte específico	
concepto de soporte auxiliar	
concepto de soporte en un soporte previo	
relación ISA	
atributo simple	
atributo de rol	
atributo referencial	
atribución univaluada	
atribución conjunto	
atribución conjunto ordenado	
atribución tupla	
atribución multi-conjunto	
atribución or	
atribución xor	
restricción de opcionalidad en atribución	
restricción de obligatoriedad en atribución	

atribución, aunque como veremos enseguida esta expresión puede ser más compleja en función del tipo de relación de atribución. Por ejemplo, en relación con la descripción del concepto fórmula que hemos mostrado más arriba (y que recordemos era fórmula∗variable simple o función de fórmulas más sencillas∗), un metamodelo que contuviera dicho concepto podría contener también un concepto función y un atributo (fórmula) DE función (aunque como veremos enseguida este atributo necesita en realidad una especificación más detallada).

Otro aspecto novedoso de la especificación de soportes en relación con atributos es que se permite establecer relaciones de especialización entre ellos, dando lugar a jerarquías ISA de atributos. El enfoque conceptual de NÓESIS es lo que fundamentalmente posibilita la existencia de esta característica. En efecto, y puesto que los atributos son también conceptos (aunque percibidos jugando un papel descriptor), las relaciones ISA son también aplicables en este caso. Sin embargo, para mantener la coherencia estructural en los soportes, es necesario imponer la siguiente restricción: se puede establecer una relación ISA entre dos atributos sólo si ambos atributos describen el mismo concepto, con la misma relación de atribución, y con las mismas restricciones de multiplicidad (las cuales serán analizadas de inmediato). Esta restricción está motivada por la observación anterior, según la cual un atributo no puede ser considerado en ningún caso de forma aislada, sino que es un atributo, de un concepto y con una determinada relación de atribución. Por ello, si un atributo es una especialización de otro, ha de serlo con la inclusión de todo lo que un atributo lleva consigo. En el momento actual, no conocemos ninguna otra técnica de metamodelización (ni incluso de modelización) que permita la aparición de relaciones de especialización entre atributos. Es bastante probable que la razón de esta situación sea que habitualmente la noción de 'atributo' en el resto de técnicas se corresponde con la noción de 'atributo simple' de la técnica NÓESIS, y en este tipo de atributos no se da en general la posibilidad de establecer especializaciones. Como veremos, la técnica NÓESIS incluye algunos otros tipos de atributos más complejos, que permiten el desarrollo de jerarquías ISA bajo ellos.

Antes de entrar en el detalle de los distintos tipos de relaciones de atribución y de atributos, es necesario hacer notar también que los atributos de un soporte satisfacen una propiedad de herencia, según la cual cada atributo es heredado por todos los conceptos que sean más especializados (dentro de la jerarquía ISA de conceptos) que el concepto al que el atributo describe de manera directa. Esta propiedad, ligada

al hecho de que un soporte es un grafo dirigido acíclico, tiene como consecuencia que la herencia de atributos pueda ser múltiple, es decir, que un concepto puede heredar atributos de todos aquellos conceptos de los que sea especialización. Al igual que otros autores (como por ejemplo [RBP+91]) entendemos que si un mismo atributo es heredado por un concepto a través de más de un camino, dicho atributo es heredado sólo una vez. Relacionado con este hecho, creemos que la herencia múltiple de atributos que resulten ser sinónimos en el concepto que hereda debe ser evitada por el meta–analista, introduciendo cambios en el soporte en caso que sea necesario.

Vamos a describir a continuación en primer lugar los distintos tipos de relaciones de atribución y en segundo lugar los distintos tipos de atributos que se distinguen en la técnica NÓESIS. Las relaciones de atribución se clasifican en primera instancia en dos grandes grupos: la atribución univaluada y la atribución multivaluada. La relación de atribución es *univaluada* cuando el 'algo' que describe (que atributa) se percibe como una única cosa, y es *multivaluada* cuando ese 'algo' se percibe como una multiplicidad de cosas. En este último caso, se consideran como posibles las siguientes atribuciones pertenecientes a estructuras matemáticas habituales: *conjunto*, *conjunto ordenado*, *multiconjunto* (conjunto con repetición) y *multiconjunto ordenado*, al que por brevedad llamaremos *tupla*. En todos los casos, las atribuciones multivaluadas se entienden homogéneas. Esto quiere decir que todos los elementos de las estructuras matemáticas asociadas deben ser instancias del mismo concepto, al que llamamos *elemento de atributo* o por brevedad *e–atributo*. Aclararemos el significado de esta idea con el ejemplo que ya hemos utilizado anteriormente respecto a los conceptos de fórmula y función. En concreto se ha dicho que para describir el concepto función podría aparecer un atributo (fórmula) DE función, ya que una función toma fórmulas como argumento. Observemos que estamos usando el plural –fórmulas–, luego en realidad esta atribución debe ser multivaluada. En concreto, los argumentos de una función forman un conjunto ordenado y con repetición, luego la atribución multivaluada que es adecuado utilizar es la de tipo tupla. Utilizaremos la notación TUPLA_ DE{ } para representar en modo textual esta relación de atribución. Por tanto el atributo sería TUPLA_ DE{(fórmula)} DE función, y lo que con anterioridad habíamos identificado como atributo, (fórmula) DE función, es en realidad el elemento de atributo (e–atributo), es decir, aquello que identifica cada uno de los elementos homogéneos de la atribución multivaluada (en el ejemplo, cada una de las fórmulas que forman parte del argumento de la función). Es necesario

notar que en el caso de atribución univaluada, las nociones de atributo y e–atributo se identifican.

En el cuadro 2.2 se muestra la representación gráfica que se utiliza tanto para la atribución univaluada (una linea simple) como para las diferentes atribuciones multivaluadas (en las que se superpone un círculo sobre una línea simple, y donde el círculo contiene un símbolo textual que es diferente en función del tipo de relación multivaluada de que se trate). Cuando la relación es multivaluada, el número de instancias del atributo que pueden describir a una misma instancia del concepto atributado puede restringirse indicando los números mínimo y/o máximo de instancias posibles. Gráficamente esta situación se representa mediante una etiqueta con el formato *(min,max)*, una notación bastante habitual para la representación de restricciones de cardinalidad.

Una segunda clasificación de relaciones de atribución está relacionada con su situación respecto a los tradicionales operadores lógicos (AND, OR, XOR). Por defecto, cuando un concepto es descrito por varios atributos se está adoptando de forma implícita la asunción del operador lógico AND. Por ejemplo en la afirmación 'el concepto C está descrito por A y B', el uso de la conjunción copulativa 'y' implica que en efecto una instancia del concepto C está descrita por una instancia del atributo A y (es decir, AND) por una instancia del atributo B[5]. Sin embargo, esta no es la única situación posible. La técnica Nóesis permite que un concepto esté descrito o bien por un atributo o bien por otro, o incluso por ambos, en función de si el operador lógico utilizado es el OR exclusivo (XOR) o no. Por ejemplo, si un concepto C está descrito por A OR B, significa que una instancia de C puede estar descrita o sólo por una instancia de A, o sólo por una instancia de B o bien por una instancia de A y una instancia de B. Por el contrario, si C está descrito por A XOR B, una instancia de C está descrita o bien por una instancia de A o bien por una instancia de B. En el cuadro 2.2 se muestra la notación gráfica que es utilizada para representar estas relaciones de atribución particulares. Por último, las relaciones de atribución pueden estar además afectadas por restricciones de obligatoriedad u opcionalidad. Estas restricciones indican si para la descripción de una instancia del concepto atributado es obligatoria (u opcional) la inclusión de instancia(s) del

[5]Es importante remarcar que cuando se dice 'una instancia del atributo A (o B)' esta instancia puede ser en realidad una multiplicidad de cosas, en función de si la relación de atribución es o no multivaluada.

atributo. El uso conjunto de las restricciones de obligatoriedad y opcionalidad y las relaciones de atribución de tipo OR y XOR ofrecen una variedad altamente expresiva de tipos de atribución. La representación gráfica de las restricciones de obligatoriedad y opcionalidad en los soportes sigue la tradicional notación de un pequeño circulo junto al concepto atributado y al principio de la línea simple que forma parte de la atribución, como se muestra en el cuadro 2.2.

Para acabar con la descripción de la noción de soporte, vamos a detallar los tipos de atributo que se distinguen. En un soporte se pueden utilizar tres tipos diferentes de atributo: simple, de rol y referencial. Un *atributo simple* es un atributo tal que su e–atributo es un concepto cuyas instancias son valores de un dominio atómico fijado. Por tanto, los atributos simples se corresponden con la noción más habitual de atributo de otras técnicas de modelización y metamodelización. Como se muestra en el cuadro 2.2, los atributos simples se representan gráficamente por medio de elipses, indicando en su interior el nombre del concepto que desempeña el papel de atributo.

En segundo lugar, en un soporte pueden aparecer *atributos de rol*. Estos atributos pueden ser utilizados fundamentalmente por dos motivos. Por una parte, estos atributos permiten el refinamiento progresivo de estructuras especificadas con poco nivel de detalle hacia estructuras más detalladas. Por otra parte, un atributo de rol (o su e–atributo) puede representar el significado del enlace entre dos conceptos, es decir, puede representar el rol que un concepto desempeña en una atribución. Dentro del ejemplo del concepto fórmula que estamos manejando podemos especificar también un ejemplo de atributo de rol. Recordemos que los argumentos de una función conforman un conjunto ordenado de fórmulas. Antes hemos representado esta situación a través del atributo TUPLA_ DE{(fórmula)} DE función. Sin embargo, para facilitar la comprensibilidad del soporte, puede ser conveniente hacer explícito el hecho de que los elementos de este conjunto desempeñan el papel de argumento. En este caso, consideraríamos el atributo de rol TUPLA_ DE{(argumento)} DE función, conjuntamente con el atributo (fórmula) DE argumento (la representación textual de estas nociones puede ser difícil de leer: en la figura 2.2 aparece la representación gráfica, que sin duda mejora la legibilidad).

En tercer y último lugar, un *atributo referencial* puede utilizarse cuando el meta–analista percibe que los individuos del e–atributo de un atributo A *siempre* pueden representarse de la misma forma que los individuos de otro concepto C. Si se proporciona una forma de representar los individuos de este último concepto, entonces

el atributo A puede ser considerado atributo referencial añadiéndole una referencia, llamada *referencia estructural*, al concepto C, que es llamado *concepto referido*. Gráficamente, tal y como se muestra en el cuadro 2.2, los atributos referenciales se representan por medio de un doble rectángulo en el interior de una elipse. Una característica importante de los atributos referenciales es que éstos y los conceptos referidos por ellos no son enlazados gráficamente de forma explícita, es decir, que no se traza ninguna línea ni ningún otro elemento gráfico entre un atributo referencial y su concepto referido. Este hecho aparentemente simple tiene gran relevancia a la hora de analizar la legibilidad de los soportes, como mostraremos algo más adelante. Opcionalmente, la técnica NÓESIS permite utilizar una etiqueta (que en la notación gráfica debe situarse junto al atributo referencial) para expresar información que se considere necesaria para localizar el concepto referido. Esta etiqueta será diferente en función del tipo de referencia estructural que se establezca.

En concreto, al establecer una referencia estructural, se pueden presentar cuatro situaciones: (1) que tanto el concepto referencial como el referido pertenezcan al mismo soporte; (2) que pertenezcan a distintos soportes; (3) que el concepto referido pertenezca al sistema de referencia (pero no pertenezca a ningún soporte); y (4) que el concepto referido *sea* un soporte. En principio no se utiliza ninguna etiqueta específica para los casos (1) y (2) (aunque como veremos de inmediato sí se utilizará etiqueta en el caso de que se imponga una restricción adicional de tipo extensional). La situación correspondiente al caso (3) facilita de forma importante la especificación de los soportes, puesto que permite que aquella información que no se considera esencial para la comprensión del soporte no tenga que ser incluida en el mismo. Sin embargo, en un soporte pueden aparecer 'llamadas' al sistema de referencia del metamodelo mediante la inclusión de atributos referenciales de este tipo, de tal forma que el analista disponga en todo momento de referencias que le permitan comprender mejor el significado de determinados conceptos. En este caso (3) junto al atributo referencial se añade la etiqueta sref. En el último caso (caso (4)), junto al atributo referencial se utiliza la etiqueta sop. La idea básica en este último caso es que las instancias del e–atributo del atributo referencial son modelos (en general complejos) que se obtienen mediante instancias de los conceptos fuente de un (otro o el mismo) soporte. Esta característica permite, por una parte, una gran flexibilidad en la organización modular de los soportes (recordemos que el marco representacional contiene un conjunto de soportes), y por otra, la expresión sencilla

de estructuras de carácter esencialmente recursivo que serían mucho más difíciles de expresar de otro modo (un ejemplo concreto de esta última situación aparece en [DZ]).

Como caso particular de referencia estructural de un atributo referencial AR a un concepto referido CR, se puede establecer una restricción extensional de tal forma que cada individuo del e–atributo de AR representado en un modelo deba ser también un individuo del concepto CR representado en el modelo. En los casos en que imponga esta restricción, se dice que AR *refiere extensionalmente a* CR, y se utiliza la etiqueta ext junto al atributo referencial.

Tal y como hemos comentado, en la representación de un soporte en forma de diagrama no se traza ningún elemento gráfico entre los atributos referenciales y sus conceptos referidos. Por otra parte, los atributos referenciales son los únicos que tienen una cierta conexión con otros conceptos que no sean aquellos directamente atributados. Es decir, los atributos simples y los de rol no están conectados de forma directa con ningún concepto excepto con el concepto atributado por cada atributo. Estas dos características, unidas a la ya establecida de que un soporte es un grafo conexo dirigido acíclico con un sólo sumidero, implican que un soporte tiene una apariencia visual muy similar a la de un árbol con raíz. Observemos que en efecto un soporte no es un árbol puesto que la dirección de los arcos es la inversa: en un árbol la dirección de los arcos es de la raíz hacia las hojas, mientras que en un soporte existe un sumidero al que 'convergen' todos los arcos del grafo. Sin embargo dentro de la técnica Nóesis habitualmente también llamamos raíz al sumidero de un soporte, en parte por la similitud con un árbol y también porque el sumidero es el concepto de soporte más genérico posible, y por tanto desempeña un cierto papel de raíz, de base, de todo el conocimiento incluido en el soporte. Esta característica tiene una repercusión mayor de la que podría deducirse en primera instancia, ya que lo que la estructura arbórea proporciona es una guía que mejora la legibilidad y comprensibilidad de un soporte, y por tanto del método representado en él. En efecto, la raíz–sumidero de un soporte puede ser utilizada como punto de partida para la lectura del mismo, de tal forma que la estructura en forma de árbol permite profundizar en el conocimiento detallado del método representado sin más que descender, hasta el nivel deseado en cada momento, por la jerarquía de conceptos. En el momento actual no conocemos ningún otro enfoque de metamodelización que proponga una representación gráfica similar a la que proporciona la técnica Nóesis,

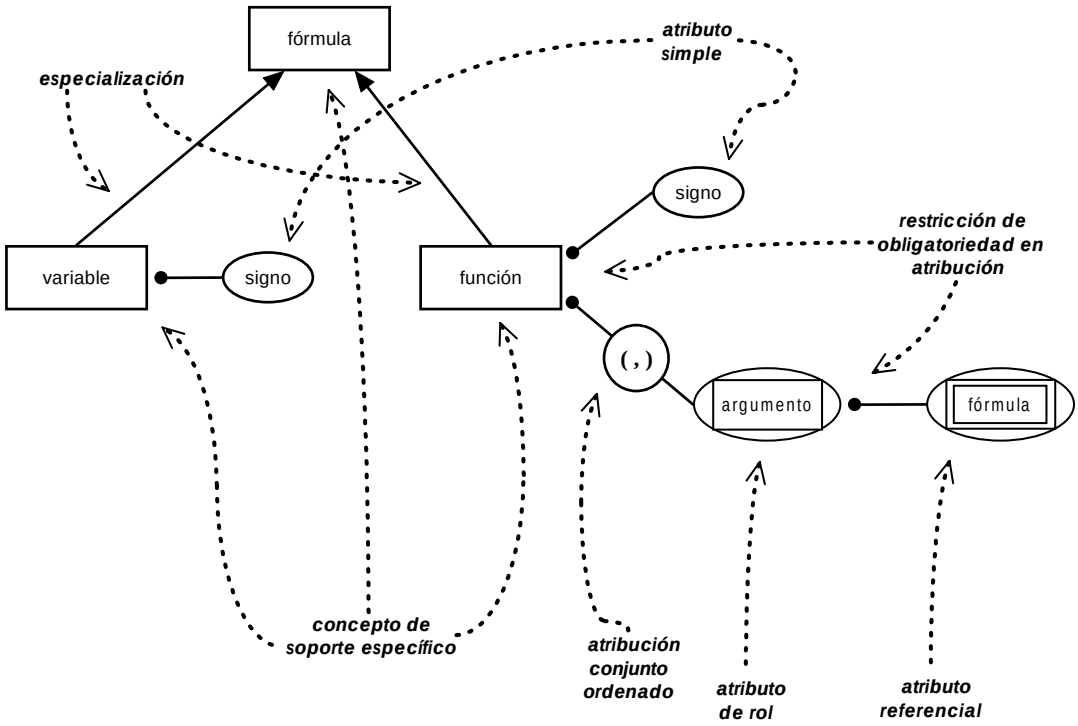


Figura 2.2: Ejemplo de soporte. Soporte de fórmulas

puesto que todas las técnicas existentes proponen representaciones en forma de grafo (véanse, por ejemplo, [HSB97], [tHvdW93] o [KLR96]) y ninguna de ellas proporciona este tipo de ayuda gráfica que guíe al analista en el aprendizaje y uso del metamodelo.

A modo de resumen de todo lo dicho sobre la noción de soporte, vamos a revisar brevemente el fragmento de soporte de la figura 2.2. En primer lugar es necesario advertir que en efecto el ejemplo presentado debe ser entendido como un *fragmento* de un soporte, que se muestra exclusivamente por motivos de exposición. Esto significa que un diagrama como el mostrado será en general parte de un soporte, puesto que la información que se recoge en la figura 2.2 es probablemente insuficiente para ser un soporte por sí misma. En segundo lugar hacemos notar que en esta figura se han incluido una serie de etiquetas con diferente tipo de letra y flechas de trazo discontinuo que no forman parte del soporte, sino que se han añadido con la intención exclusiva de que sea más fácil identificar los distintos tipos de elementos constitutivos de un soporte. Dicho esto, observemos que este fragmento de soporte corresponde esencialmente con la descripción del concepto fórmula que estamos utili-

zando como ejemplo en esta sub-sección. En este ejemplo de (fragmento de) soporte aparece la mayoría de los elementos que hemos venido presentando. En concreto, los conceptos fórmula, variable y función son conceptos específicos de soporte, y de tal forma que se establecen sendas relaciones de especialización entre el concepto fórmula y los otros dos conceptos específicos de soporte. El concepto variable aparece descrito por un atributo (signo) DE variable y el concepto función aparece descrito por el atributo (signo) DE función y otro atributo adicional, que ya ha sido descrito con anterioridad. Este atributo es un atributo de rol con atribución multivaluada de tipo conjunto ordenado, de tal forma que el e–atributo es (argumento) DE función y el atributo en sí es TUPLA_ DE{(argumento)} DE función. A su vez, el concepto argumento aparece atributado por un atributo referencial con atribución univaluada, el atributo (fórmula) DE argumento. Este atributo referencial tiene como concepto referido el concepto de soporte específico fórmula dentro del mismo soporte. Por último, observemos que en este fragmento de soporte todas las atribuciones están afectadas por la restricción de obligatoriedad.

En referencia a la noción de soporte es necesario realizar una última observación. Recordemos que un marco representacional incluye (junto a un conjunto de restricciones locales) un conjunto de soportes. Esta característica es una novedad que presenta la última versión de la técnica NÓESIS [DZ] respecto a la primera [DZR97], y permite alcanzar un alto grado de modularidad. Si bien es cierto que en muchos casos un marco representacional contendrá un único soporte, la posibilidad de incluir un número mayor permite estructurar un soporte que se considere demasiado complejo en varios soportes más sencillos, sin pérdida de legibilidad ni conexión entre ellos gracias a la existencia de los atributos referenciales. De igual forma, esta característica hace que los metamodelos NÓESIS estén perfectamente adaptados a las necesidades de la ingeniería situacional del método, ya que cada fragmento de método puede representarse a través de un soporte. La tarea del ensamblaje de métodos puede ser más sencilla de realizar estableciendo las correspondientes relaciones entre los soportes por medio de atributos referenciales, y de tal forma que todos los soportes, considerados en conjunto, formen parte de un único metamodelo (nuevo método *ad hoc*) resultado del ensamblaje de métodos. Un ejemplo parcial de esta situación se muestra en [DZ], y mostraremos en la siguiente sub-sección otras características de la técnica NÓESIS diseñadas como apoyo a las tareas de la ingeniería situacional del método.

Definición de Modelo

La última componente de cada metamodelo NÓESIS es la *componente de definición de modelo*. En esta componente debe definirse qué es exactamente un modelo del método que se está representando con el metamodelo. Utilizando la terminología del Cubo de Hofstede y Verhoef, lo que esta componente proporciona es la definición de la conceptualización de los aspectos de producto en el Nivel de Aplicación, a partir de la información recogida en el Nivel del Método. En particular observemos que la definición de modelo no tiene porqué incluir en general los aspectos de representación (recordemos que en la figura 2.1 la única dimensión de modelización que aparece sombreada en el Nivel de Aplicación es la del aspecto producto a nivel conceptual), ya que un metamodelo determina cuáles son los conceptos utilizados para crear los modelos, pero un mismo concepto puede tener asociadas distintas notaciones. En la primera versión de la técnica NÓESIS [DZR97] se introduce el término *metamodelo comunicable* para referirse a un metamodelo al que se le hayan añadido reglas concretas de representación para sus modelos.

En general, un modelo (del Nivel de Aplicación) de un metamodelo NÓESIS se construye a partir de instancias de los conceptos fuente del soporte. Pero además, dentro de la definición de modelo se pueden incluir *restricciones globales*, que son condiciones que un modelo, de forma global, debe cumplir (un ejemplo de conjunto de restricciones globales se muestra en la página 121 para el metamodelo NÓESIS de RM/T). Observemos que el papel de estas restricciones globales es diferente del que tenían las restricciones locales que se incluyen en el marco representacional. Mientras que éstas últimas han de ser tenidas en cuenta durante el proceso de construcción de los modelos, afectan sólo a partes determinadas de los mismos y pueden abordarse en general de forma individual, separada del resto, las restricciones globales deben ser verificadas sólo en el último paso de la construcción del modelo, considerando éste en su conjunto. De nuevo, en el momento actual no conocemos ninguna técnica de modelización que realice esta distinción entre restricciones locales y globales. En cuanto a la notación, las restricciones globales se expresan de igual manera que las locales, por medio de lenguaje natural.

2.1.2. Otras características de la técnica Nóesis

Como hemos observado en los comentarios iniciales de esta sección, aunque en principio la técnica NÓESIS no se ocupa en el momento actual del aspecto 'proceso' del Nivel del Método, sí que proporciona ciertos elementos relacionados con este aspecto, y que en particular tienen que ver con el uso de la técnica como herramienta de ingeniería situacional del método.

Recordemos que durante el desarrollo de un sistema de información los métodos estándar no son utilizados exactamente tal y como vienen definidos. Las circunstancias particulares de cada proyecto hacen que sea necesario adaptar los métodos para poder manejar adecuadamente cada situación que se presente [CG94, GV96, Ode96]. Este es el ámbito de la ingeniería situacional del método [Har97], en la que el término *método situacional* es utilizado para referir a un método ajustado a las necesidades de un marco de desarrollo particular. Un método situacional puede ser obtenido ensamblando de forma coherente, fragmentos de otros métodos, lo llamados *fragmentos de método*, haciendo uso de un entorno de Ingeniería del Método Asistida por Ordenador (Computer Aided Method Engineering – CAME) el cual proporciona operaciones para diseñar, almacenar, recuperar y ensamblar fragmentos de método [Bri96, HBO94]. Con respecto al proceso de ensamblaje, en [tHV97] se distinguen dos tareas diferentes: (1) la (posible) adaptación de los fragmentos de método almacenados y (2) la integración de los fragmentos resultantes.

En relación con la técnica NÓESIS, puesto que trata de ser una técnica adecuada para la ingeniería situacional del método, ha de estar provista de mecanismos que permitan la manipulación y modificación de sus metamodelos. Como un primer paso para ello, en [DZ] se analizan los mecanismos que son necesarios para la modificación de los soportes. Teniendo en cuenta que un soporte puede ser visto como un fragmento de método, estos mecanismos permiten al meta–analista llevar a cabo procesos de adaptación de fragmentos de método. Además, la aparición de los atributos referenciales discutidos con anterioridad hacen también pueda ser llevado a cabo un cierto tipo de integración de métodos.

Tal y como se afirma en [tHV97], para llevar a cabo las tareas de adaptación de métodos, un entorno CAME debería proporcionar a los ingenieros del método transformaciones de metamodelos adecuadas, las cuales deben verificar propiedades tales como correción, completitud y consistencia. De forma similar, el enfoque que la

técnica NÓESIS propone para la adaptación de métodos esta basada en la definición de una familia de transformaciones que verifican las propiedades de minimalidad y completitud. Esta familia minimal y completa es llamada *base de transformaciones* y cada una de las transformaciones en ella es llamada *transformación primitiva*. La idea es que cualquier proceso de adaptación puede ser llevado a cabo por medio de una composición de transformaciones primitivas. Estas transformaciones, así como el detalle de su descripción y utilización se explican extensamente en [DZ]. Además de esto, dentro del marco de la técnica NÓESIS, en [DZ00] se ha analizado el uso de transformaciones en el contexto de la interoperabilidad de métodos.

2.2. Un metamodelo Nóesis de RM/T

Una vez descritas en la sección anterior las características esenciales de la técnica NÓESIS de manera fundamentalmente teórica, vamos a presentar ahora un ejemplo práctico de aplicación de la técnica. En concreto, y como ya ha sido adelantado al comienzo del capítulo, el ejemplo que vamos a presentar consiste en un metamodelo detallado del modelo de bases de datos RM/T. Antes de mostrar específicamente este metamodelo NÓESIS vamos a presentar brevemente las distintas referencias que sobre dicho modelo podemos encontrar en la literatura.

2.2.1. Referencias sobre RM/T

El modelo de bases de datos RM/T fue presentado originalmente por E.F. Codd en el año 1979 en el transcurso de la reunión de la Australian Computer Society celebrada en Hobart, Tasmania. Allí, Codd presentó el artículo "Extending the Database Relational Model to Capture More Meaning", y la extensión del modelo relacional que se proponía en este artículo recibió el nombre de RM/T (Relational Model/Tasmania) en honor al lugar donde fue presentado por vez primera. Posteriormente, este artículo fue publicado en la revista ACM Transactions on Database Systems [Cod79]. Esta es sin duda "la referencia" para tratar sobre RM/T, y por tanto es la fuente que hemos utilizado fundamentalmente para la realización de su metamodelo. Las otras dos referencias básicas sobre RM/T son los capítulos correspondientes de sendos libros de C.J. Date: el capítulo 6 del volumen II de "Introduction to Database Systems" [Dat85] y el capítulo 7 de "Relational Database

Writings: 1989–1991" [Dat92].

El objetivo que Codd se proponía con esta extensión del modelo relacional se muestra de forma clara en el abstract de [Cod79]:

> "Durante los últimos tres o cuatro años diversos investigadores han estado examinando «modelos semánticos» para bases de datos formateadas[6]. La intención es la de capturar (de manera más o menos formal) más significado de los datos, de tal forma que el diseño de bases de datos pueda ser más sistemático y el sistema de bases de datos en sí pueda comportarse de forma más inteligente. Dos ideas centrales son claras:
>
> 1. la búsqueda de unidades con significado tan pequeñas como sea posible – semántica atómica;
> 2. la búsqueda de unidades con significado que sean mayores que la relación n-aria habitual – semántica molecular.
>
> En este artículo proponemos extensiones del modelo relacional que sirven de fundamento a cierta semántica atómica y molecular.[...]"

Por tanto, lo que Codd pretendía con el modelo RM/T era un avance en el diseño y uso de las bases de datos, de tal forma que estas pudieran expresar mejor el significado de los datos en ellas contenidos. Este tipo de enfoque era y es habitualmente referido por el término, sin duda impreciso, de *modelo semántico*[7]. De hecho, diversas referencias en la literatura incluyen al modelo RM/T dentro de la categoría de modelos semánticos (véanse por ejemplo [HK87, PM88, Ros84, Sta85, STW84]), categoría en la que la gran mayoría de las referencias incluyen también al modelo

[6]El término *base de datos formateada* puede resultar chocante en la actualidad, pero no debe perderse de vista que este artículo se escribió en el año 1979, menos de 10 años después de la aparición del artículo original de definición del modelo relacional [Cod70]. Por tanto en aquellos años la idea (asumida hoy en día) de que una base de datos ha de tener una estructura –un formato– definido y bien especificado, no se encontraba extendida de forma general. El concepto de base de datos formateada está descrito en [Cod79, p. 408].

[7]Es llamativo el hecho de que Codd rehuyera de utilizar en el título de su artículo la palabra 'semántica' (usando el término 'significado') y que en el abstract pusiera entre comillas el término "modelos semánticos", reservando el uso del término 'semántica' para referirse a aspectos muy concretos de su propuesta.

E/R de Chen [Che76]. El término 'semántica' es sin lugar a dudas un término enormemente sobrecargado en el ámbito de las Ciencias de la Computación. Esto ha sido hecho notar muy recientemente en referencia a UML, por ejemplo en [KER99], donde se dice en tono jocoso que "hoy en día, existe un alto grado de confusión debido al hecho de que la palabra *semántica* tiene muchas semánticas diferentes". En el caso de los modelos semánticos de bases de datos, posiblemente la elección de Codd de utilizar el término 'significado' en lugar 'semántica' sea correcta. Esto es así puesto que cuando se habla de modelo semánticos, se quiere hacer referencia por una parte al hecho de que este tipo de modelos proporciona modos de estructurar los datos que son más ricos que los que proporcionan los modelos "tradicionales", y por otra a la mayor o menor adaptación que tiene ese modelo para expresar con facilidad la información del mundo real que una base de datos recoge. Uno de propósitos fundamentales (quizá el más fundamental) de una base de datos es que la representación que (de una parte del mundo real) supone la base de datos sea una representación fiel, coherente con el mundo representado. Cuando un modelo de bases de datos proporciona herramientas que acercan en cierta medida el mundo real al mundo de la bases de datos, de tal manera que la tarea de representación se hace más fácil para el diseñador, y los usuarios finales ven reflejados con mayor claridad las ideas del mundo real en la base de datos, es cuando se habla de que un modelo es un modelo semántico. Esto es expresado claramente por ejemplo en [HK87]: "Los modelos semánticos han sido desarrollados para proporcionar un mayor nivel de abstracción para la modelización de datos, permitiendo a los diseñadores de bases de datos pensar en los datos en modos que se correlacionan más directamente con la forma en que los datos aparecen en el mundo". En este sentido es en el que se puede entender el modelo RM/T como un modelo semántico, pues en efecto, como mostraremos, proporciona ciertos elementos que intentan acercar más los datos al significado que tienen en el mundo real.

El modelo RM/T, no obstante, no ha tenido prácticamente aceptación entre los desarrolladores de Sistemas Gestores de Bases de Datos (SGBD). Una de las razones fundamentales de esta situación es expuesta por el propio Codd en el prefacio de "The relational model for database management, version 2" [Cod90]. En este libro Codd se refiere a la 'primera versión del modelo relacional' mediante las siglas RM/V1, considerando que ésta corresponde al periodo comprendido entre la aparición del primer artículo motivador del modelo [Cod69] hasta el artículo de definición

del RM/T, y de tal forma que en el libro se define la 'segunda versión del modelo relacional' o RM/V2. Pues bien, como decíamos, en este libro Codd reconoce una posible razón para el escaso éxito de RM/T, afirmando que "los vendedores de productos SGBD en muchos casos no han comprendido la primera versión RM/V1", y que difícilmente podrían por tanto desarrollar un sistema gestor basado en RM/T. Aparentemente, el proyecto de Codd consiste en proponer diferentes versiones sucesivas del modelo relacional (RM/V1, RM/V2, RM/V3, etc.) de tal forma que cada una de ellas recoja progresivamente las características avanzadas propias de RM/T. En el momento actual, sin embargo, la última versión del modelo relacional es la ya citada versión 2 (RM/V2).

Otra posible razón por la que el modelo RM/T no ha sido más ampliamente utilizado es expuesta por C. J. Date en [Dat92]: cuando apareció RM/T en 1979, ya se había establecido firmemente entre los profesionales de las bases de datos el modelo E/R (presentado originalmente en el año 1976 [Che76]), y la extensa investigación desarrollada en torno a este modelo (con multitud de revisiones, extensiones y mejoras) desplazó el interés hacia este ámbito. Sin embargo sí podemos encontrar en la literatura distintas referencias que tienen que ver con RM/T, si bien con enfoques bastante heterogéneos.

Así por ejemplo podemos encontrar ejemplos de trabajos que utilizan RM/T como soporte, tales como [EKTW87] sobre prototipado rápido usando RM/T; [KCL87] sobre manejo de valores nulos en bases de datos; algunos cuyo eje central son investigaciones médicas, tales como [Win87] o [Lev90]; o más recientemente [Ozk95] sobre el uso de RM/T para la definición de estructuras de búsqueda para la recuperación de documentos multimedia en bases de datos. Otro estilo de investigación sobre RM/T es la que profundiza en el desarrollo del propio modelo, en el que podemos encontrar artículos tales como [RM83] que desarrolla un cierto tipo primitivo de 'metamodelo' de RM/T utilizando como lenguaje el propio RM/T; [RP91] que añade soporte temporal al modelo o [Sal87] que trata aspectos de relativismo semántico en el contexto de RM/T. Un caso especialmente relevante de investigación que profundiza en el desarrollo de RM/T es el que se muestra en [Wee91]. En este trabajo se presenta una extensión del modelo RM/T desarrollada bajo el paradigma de la orientación a objeto. Esta extensión, denominada *RMT/OO*, fue probablemente la precursora de otra, denominada *RM/T++* presentada en [EW91]. Esta extensión fue utilizada en uno de los pocos proyectos conocidos de desarrollo de sistemas software en los que

RM/T ha sido tomado como motor de la base de información, el proyecto Aspect [Bro91], en el que se desarrollaba un Entorno Integrado de Soporte de Proyecto (Integrated Project Support Environment-IPSE). Por otra parte, el modelo RM/T tiene, tal y como debe ser en el desarrollo de cualquier propuesta científica, detractores y defensores. Saltor en [Sal81] realiza una crítica razonada del modelo, crítica que es contestada por Date en [Dat85]. Precisamente el propio Date ha continuado defendiendo en fechas recientes el modelo RM/T [Dat99].

2.2.2. Un metamodelo Nóesis de RM/T: parte intensional básica

Una vez revisadas diferentes referencias que podemos encontrar en la literatura en torno a RM/T, vamos a comenzar el desarrollo del metamodelo de este modelo de base de datos, utilizando la técnica NÓESIS como técnica de metamodelización. Es importante hacer notar que, puesto que este ejemplo se va a mostrar con la intención de demostrar la potencia expresiva la técnica NÓESIS, vamos a mostrar el ejemplo con un nivel muy alto de detalle. En particular esto va a implicar que en el desarrollo vamos a utilizar *ad hoc* algunas de las herramientas básicas de la técnica. Más en concreto, las descripciones de conceptos que deberían estar recogidas en el sistema de referencia del metamodelo vamos a mostrarlas de forma progresiva conforme vayan siendo necesarias durante el desarrollo, aunque se incluirán también en el formato habitual de tabla del sistema de referencia. De igual modo, vamos a explotar las propiedades de la componente marco representacional, que recordemos incluye de forma genérica un conjunto de soportes. En el caso concreto del metamodelo que presentamos, incluiremos dentro de este conjunto ciertos diagramas que si bien no se corresponden de forma totalmente exacta con la noción de soporte NÓESIS, entendemos que son útiles durante la exposición del metamodelo. A estos diagramas los llamaremos *soportes parciales* o *sub-soportes*.

La primera componente de todo metamodelo NÓESIS es la perspectiva, que nos proporciona una panorámica general del lenguaje que se presenta en el metamodelo. La perspectiva del metamodelo NÓESIS de RM/T se muestra en la tabla 2.3.

Tal y como se muestra en el título de esta sub–sección, comenzaremos con la descripción de la parte intensional del modelo RM/T. Si bien es cierto que en la definición del modelo no se hace (ni tan siquiera en la del modelo relacional básico)

Tabla 2.3: Perspectiva del metamodelo NÓESIS de RM/T

PERSPECTIVA DE RM/T
El modelo RM/T es un modelo de bases de datos ideado por E.F. Codd como una extensión del modelo relacional clásico. El objetivo del modelo es el de capturar más información del significado de los datos, lo cual se consigue mediante la introducción de diversos conceptos y estructuras. En primer lugar, el modelo RM/T recoge la idea de que una base de datos almacena información sobre entidades del mundo real. En la base de datos estas entidades se representan mediante identificadores únicos, asignados por el sistema, llamados *surrogates*. Además, se distinguen distintos tipos de relaciones (tablas) en el modelo, en particular *E-relaciones* y *P-relaciones*. Cada E-relación almacena los surrogates de cada tipo de entidad concreto, y por tanto se utilizan para registrar la *existencia* de estas entidades. Por su parte cada P-relación almacena algunas propiedades de las entidades registradas en la E-relación correspondiente. Por otra parte, el modelo RM/T establece un sistema de clasificación de tipos de entidades, de tal forma que estas pueden ser *característica* (si desempeña un papel subordinado describiendo otra entidad), *asociativa* (si desempeña un papel superior interrelacionando otras entidades) o *núcleo* (si no desempeña ninguno de los papeles anteriores). Esta clasificación se complementa con la posibilidad de especificar que una entidad sea *subtipo* de otra, y la estructuración de las posibles jerarquías de entidades mediante *generalización incondicional*, *generalización alternativa* o *agregación cover*. Por último el modelo incluye la descripción de un conjunto de operadores que permiten la manipulación de las estructuras anteriores.

una separación estricta, formal, entre parte intensional y parte extensional, actualmente esta forma de abordar el modelo relacional está ampliamente admitida en la literatura (véanse por ejemplo [ACPT99, EN02, UW02]).

Fundamentos de la parte intensional del modelo RM/T

A continuación iremos mostrando uno a uno, los distintos conceptos que aparecen en el modelo RM/T, utilizando la sintaxis que se ha explicado en la sección 2.1.1. En la mayoría de los casos cada concepto se acompaña de una breve explicación, en lenguaje natural, del concepto en cuestión, incluyendo en su caso ejemplos aclaratorios.

dominio ∗conjunto de valores de tipo similar∗	[Cod79, p.399, sec 2.1]

Para la técnica NÓESIS, nos encontramos ante un concepto epistemológico, es decir, que no se deduce del resto de conceptos del metamodelo y que recurre al conocimiento común para su comprensión. En este caso concreto la descripción del concepto dominio es bastante imprecisa, ya que resulta ambigua la noción de 'tipo similar': un dominio podría ser la lista $\{a, e, i, o, u\}$ (considerando que el tipo similar es que son letras vocales) pero también $\{coche, naranja\}$ (considerando que el tipo es que son palabras). Sin embargo, esta es la única descripción utilizada por Codd.

nombre_de_dominio ∗∗	[Cod79, p.424, sec 14] int.

En general no todos los dominios tienen porque estar nominados. Así, podemos dar el nombre 'vocales' al dominio $\{a, e, i, o, u\}$, pero el dominio $\{coche, naranja\}$ puede no tener nombre. En este último caso para referir al dominio hay que hacerlo mediante la expresión explícita del mismo (su conjunto de valores, que a su vez también puede estar expresado de forma implícita, como en el dominio $\{n; 0 \leq n \leq 120\}$). El concepto nombre_de_dominio es un ejemplo de concepto no–descrito dentro del metamodelo.

atributo ∗agregación de un nombre y un dominio o nombre_de_dominio∗	[Cod79, p.399, sec 2.1] int.

Esta descripción esta más cerca de la dada por Date en su revisión del modelo relacional [Dat01] que de la dada por Codd en el artículo de definición de RM/T. En este último, Codd define los atributos como *índices* para un conjunto de dominios, y no interpreta que un atributo tenga incluido un dominio sino que tiene asociado un dominio. Con la interpretación de Codd un atributo se limitaría a un nombre, lo que es en cierto modo contradictorio con la forma en que después se manipulan los atributos dentro del modelo relacional. Por otra parte, para especificar un atributo se permite dar o bien un dominio, en cuyo caso hay que interpretar que lo que se da es el conjunto de valores explícitos que constituye el dominio, o bien un nombre de dominio, en cuyo caso hay que interpretar que se usa o bien un dominio básico del sistema o bien un dominio definido (nominado) por el diseñador. Observemos que en nuestra descripción, el término 'agregación' es epistemológico, y debe entenderse con el sentido más habitual de 'poner junto'. Ejemplos de atributos bajo esta descripción podrían ser $<persona, varchar(40)>$, $<color, \{rojo, verde, azul\}>$ o

$\langle edad, \{n; 0 \leq n \leq 120\}\rangle$. En estos ejemplos estamos utilizando una notación concreta para poder expresar atributos, pero debe quedar claro que la elección de una notación particular no es relevante en este momento.

nombre_de_atributo ∗∗	[Cod79, p.424, sec 14] int.

Es importante recalcar que de estos conceptos no hay que inferir automáticamente ninguna intención de uso. Por ejemplo, cuando decimos que un atributo tiene por nombre de atributo 'edad', este nombre debe entenderse de forma esencialmente simbólica. Es cierto que cuando un atributo tal se utilice en el desarrollo de una base de datos, se utilizará con una cierta intención de uso (que será la de representar la edad de algo o de alguien), pero en este momento del desarrollo del metamodelo el nombre de un atributo simplemente debe ser entendido como la nominación del propio atributo.

dominio_de_atributo ∗∗	int.

A partir de los elementos básicos de dominio y atributo, se construyen nuevos elementos algo más complejos.

esquema_de_relación_sin_nombre ∗conjunto de atributos∗	[Cod79, p.414, sec 7], int.
esquema_de_relación_con_nombre ∗agregación de un nombre y un conjunto de atributos∗	[Cod79, p.414, sec 7], int.
nombre_de_esquema_de_relación_con_nombre ∗∗	int.
esquema_de_relación ∗esquema_de_relación_sin_nombre o esquema_de_relación_con_nombre∗	[Cod79, p.399, sec 2.1], int.

La presentación de diferentes descripciones para esquemas de relación con nombre o sin nombres está motivada por los comentarios de Codd [Cod79, p.414,sec.7] así como por la introducción de ciertos operadores [Cod79, p.425,sec.15], donde se especifica de manera clara que no toda relación (y por tanto tampoco el esquema de relación) debe tener nombre, caso por ejemplo de relaciones construidas en consultas.

Incluiremos también de forma progresiva las diferentes restricciones (que recordemos pueden ser locales, incluidas en el marco representacional, y que indicaremos con una letra 'L' delante del número de restricción; o globales, incluidas en la com-

ponente de definición de modelo, indicadas con una letra 'G' delante del número) que es necesario imponer a los modelos construidos a partir del metamodelo. Con respecto a los esquemas de relación la restricción que es necesario imponer es que:

Restricción L1	Todos los atributos de un esquema_de_relación deben tener distinto nombre_de_atributo.

[Cod79, p.399, sec 2.1] int.

Observemos que aunque en principio en la técnica NÓESIS no se exige la inclusión de referencias en la especificación de restricciones hemos creído conveniente incluirlas para permitir un mejor control de calidad del metamodelo.

A la vista de este principio, un ejemplo de esquema de relación con nombre podría ser

$$\text{<}empleado, \{DNI, nombre, edad\}\text{>}$$

Observemos que en esta notación, por simplicidad, hemos indicado cada atributo sólo a través de su nombre. Una sintaxis más rigurosa con respecto a las descripciones de conceptos previas podría ser

$$\text{<}empleado, \{\text{<}DNI, char(8)\text{>}, \text{<}nombre, varchar(40)\text{>}, \text{<}edad, \{n; 0 \leq n \leq 120\}\text{>}\}\text{>}$$

aunque claramente se pierde legibilidad. En cualquier caso recordemos que estas expresiones se están utilizando a modo de ejemplo y en ningún caso definen ninguna sintaxis concreta de los elementos del modelo.

dominio_de_esquema_de_relación ∗producto cartesiano de los dominios_de_atributo de esquema_de_relación∗	int.

Este concepto se incluye por completitud, ya que una de las formas habituales de definición del modelo relacional es a través del producto cartesiano de los dominios (es de hecho una de las nociones de partida para Codd). En nuestro metamodelo no será utilizada por el momento.

grado_de_esquema_de_relación ∗número de atributos de esquema_de_relación∗	[Cod79, p.399, sec 2.1] int.

esquema_de_relación_n-ario ∗esquema_de_relación de grado n∗	[Cod79, p.408, sec 3] int.

Atributos en RM/T

En realidad, hasta aquí todos los conceptos pertenecen en realidad al modelo relacional 'básico'. El primer concepto perteneciente de manera explícita al modelo RM/T es:

[E]-dominio ∗dominio que se elige entre un conjunto de dominios y se nomina E-dominio∗ [Cod79, p.410, sec 4] int.

Codd introduce la idea de E-dominio como un dominio especial de cada base de datos RM/T, cuyos valores son 'surrogates'. La traducción literal de este término al castellano es 'sustituto', lo que corresponde sólo parcialmente con el significado de 'surrogate'. Es fundamental incluir aquí literalmente las palabras de Codd [Cod79, p.410, sec 4]:

> "La necesidad de identificadores únicos y permanentes para las entidades de una base de datos [...] es clara. Existen [...] dificultades al utilizar claves controladas por el usuario como surrogates [sustitutos] permanentes de las entidades. [...] Una solución consiste en introducir dominios de entidad que contengan surrogates asignados por el sistema. Los usuarios de la base de datos pueden hacer que el sistema genere o borre un surrogate, pero no tienen control sobre su valor, ni su valor les será nunca mostrado. Los surrogates se comportan como si cada entidad (con independencia de su tipo) tuviera su propio surrogate permanente, único en el conjunto de la base datos. [...] Dos surrogates son iguales en el modelo relacional si y sólo si denotan a la misma entidad en el mundo percibido de entidades".

Es decir, Codd (si bien inspirado por trabajos previos, [HOT76]) introduce en las bases de datos la noción de *identificador único de objeto, OID*, básica en el paradigma de la orientación a objeto. Una vez más Codd fue pionero, puesto que aunque es posible que la noción de identificador único de objeto existiera en el ámbito de la programación orientada a objeto con anterioridad, Codd fue el primero que lo incluyó dentro del mundo de las bases de datos, siendo posteriormente un concepto utilizado también por los enfoques de bases de datos objeto/relacionales, y en general en el ámbito del análisis y diseño orientados a objeto.

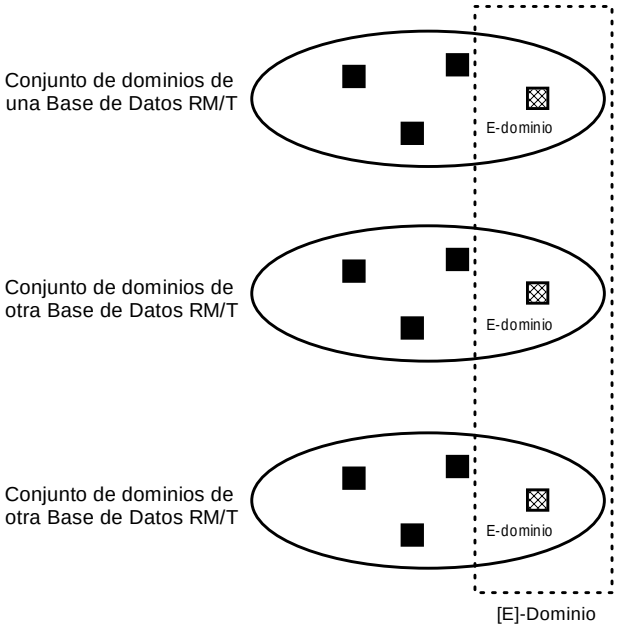


Figura 2.3: Nociones de E-dominio y [E]-dominio

Para entender mejor el concepto de [E]-dominio, sigamos citando literalmente a Codd: "En cualquier base de datos RM/T uno de los dominios básicos sirve como fuente de todos los surrogates: este dominio es llamado E-dominio". Observemos que por tanto un E-dominio pertenece a una base de datos concreta (o quizá a un SGBD concreto), pero en el metamodelo hemos de definir la noción que sirva *para cualquier modelo (base de datos) posible*, es decir, hemos de definir un *E-dominio genérico*, que es al que hemos denominado [E]-dominio. Podríamos entender el [E]-dominio como el conjunto de todos los E-dominios posibles en el conjunto de todas las bases de datos RM/T (ver figura 2.3). Date se aproxima a este planteamiento en [Dat85, p.245, sec 6.4], puesto que en él se utiliza el símbolo 'ç' como nombre genérico para los E-dominios. Esta idea está a su vez inspirada en la observación de Codd según la cual cualquier atributo definido sobre el E-dominio ha de tener un nombre acabado en 'ç' para facilitar el reconocimiento de estos atributos especiales. Profundizaremos de inmediato en este aspecto, pero previamente es necesaria la inclusión de la siguiente restricción que es en parte de tipo global, pero que en cierto modo debe ser también entendida como una meta-restricción.

Restricción G1	El [E]-dominio no es dominio

int.

Esta restricción se introduce puesto que no es correcto considerar al [E]-dominio como un conjunto de valores, y sin embargo el nombre que hemos elegido para él podría llevar a confusión. Observemos que como máximo el [E]-dominio puede ser considerado como un conjunto de dominios, con la particularidad de que todos ellos reciben el mismo nombre, E-dominio. El nombre E-dominio es un nombre concreto, un nombre propio, mientras que [E]-dominio es un nombre genérico. La restricción impuesta impide caer en paradojas como la famosa paradoja de Russell sobre el conjunto de todos los conjuntos. De alguna forma, el [E]-dominio pertenece a un nivel superior de abstracción, de ahí que la restricción quizá deba ser considerada con mayor propiedad como una meta-restricción.

E-atributo ∗atributo con dominio E-dominio∗	[Cod79, p.410, sec 4] int.

En esta descripción se ve claramente que las nociones de E-dominio y E-atributo están íntimamente ligadas, y tal y como hemos observado, Codd sugiere que los E-atributos tengan un nombre especial acabado en 'ç'. Como veremos los E-atributos jugarán el papel de clave principal asignada por el sistema, aunque esto no significa que dejen de existir claves de usuario. Codd lo muestra de manera explícita: «La introducción del E-dominio, los E-atributos y los surrogates no provoca que las claves controladas por el usuario sean obsoletas. Los usuarios a menudo necesitarán identificadores de entidad [...] que estén totalmente bajo su control, aunque ya no estarán obligados a inventar una clave controlada por el usuario si no lo desean».

P-atributo ∗atributo que no es E-atributo∗	int.

Los atributos 'tradicionales' reciben el nombre de *P-atributo*, y a la vista de la definición es obvio que todo atributo en RM/T es o bien E-atributo o bien P-atributo. Los nombres de estos tipos de atributo identifican que los E-atributos son atributos para representar **E**ntidades y que los P-atributos son atributos para representar **P**ropiedades.

En la figura 2.4 se muestra el primer ejemplo de soporte parcial, en este caso recogiendo los conceptos relativos a atributos. Un soporte parcial o sub-soporte, como ya hemos observado, no se corresponde de manera exacta con la noción de soporte de la técnica NÓESIS. Esencialmente esto es así puesto que un soporte parcial no describe un método completo, ni tan siquiera un fragmento de método, sino sólo

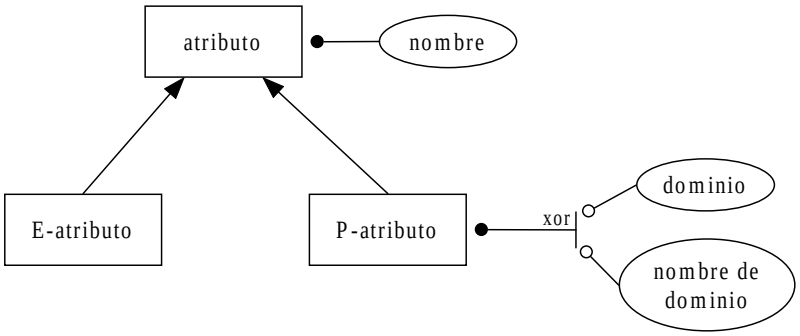


Figura 2.4: Soporte parcial. Atributos en RM/T

ciertas características o componentes de un método. Dicho de otro modo, no es posible definir un modelo a partir de un soporte parcial. Sin embargo entendemos que la inclusión de soportes parciales dentro del conjunto de soportes que a su vez forma parte del marco representacional de un metamodelo NÓESIS puede facilitar la legibilidad del metamodelo y por tanto la comprensión del lenguaje metamodelizado. En el caso particular del soporte parcial de atributos en RM/T, lo que este subsoporte muestra son los tipos de atributos permitidos en RM/T. Aunque insistimos que no es posible definir un modelo (ni RM/T ni otro) a partir de este soporte parcial, sí que se puede dar una interpretación a sus instancias. En concreto, lo que este soporte parcial implica es que *todos los atributos de un modelo RM/T serán instancias de los conceptos fuente del soporte parcial de atributos.* Esta propiedad en apariencia poco importante en este caso sencillo, será aplicable sobre los conceptos adecuados para cada soporte parcial que se incluya, e incrementará su importancia cuando los conceptos asociados sean más complejos.

Sobre el contenido del soporte parcial en sí es interesante puntualizar dos aspectos. En primer lugar, observemos que el concepto E-atributo no aparece descrito específicamente por ningún atributo NÓESIS, a excepción del atributo nombre heredado del concepto atributo[8]. Esta situación aprovecha una característica importante

[8]En esta frase se muestra muy a las claras uno de los problemas más frecuentes que surgen al tratar aspectos de modelización y metamodelización: la confusión terminológica. En este caso concreto hay que distinguir nítidamente entre la noción de atributo del modelo RM/T y la noción de atributo de la técnica NÓESIS, que si bien comparten la intención de uso de ser 'descriptores de algo', son nociones esencialmente diferentes. Por este motivo en esta frase mostramos con distintos tipos de letra ambas nociones (véanse: atributo RM/T y atributo NÓESIS), aunque sólo utilizaremos

de los soportes, que implica la no identificación de conceptos aun a pesar de que las descripciones que aparezcan en ellos sean las mismas. En este caso concreto, E-atributo es una especialización *propia* de atributo, puesto que al especificar que un atributo *es* E-atributo estamos obligando a que el dominio de dicho atributo sea el E-dominio. Como segundo aspecto destacado, observemos que la especificación de un P-atributo requiere la especificación de un dominio o (o exclusivo, xor) un nombre de dominio. Esto es así puesto que ya hemos explicado que *a priori* no todos los dominios tienen porqué ser nominados.

Esquemas de relación en RM/T

esquema_de_E-relación ∗esquema_de_relación_unario cuyo atributo es E-atributo∗	[Cod79, p.410, sec 5] int.

Este es otro de los puntos claves y novedosos del modelo RM/T. La intención de uso de este tipo de esquemas es que cada tipo entidad tenga asociado un esquema de E-relación. Dichos esquemas serán la base de la correspondiente E-relación que almacenará los surrogates correspondientes a cada instancia de entidad [Cod79, p.410–411, sec 5]. El propio Codd introduce la siguiente restricción:

Restricción L2	El nombre_de_E-atributo de esquema_de_E-relación es el mismo nombre que el del esquema añadiendo al final el carácter 'ç'.

[Cod79, p.410, sec 5] int.

Observemos en primer lugar, respecto a esta restricción, como distintos conceptos que se han ido incluyendo en el sistema de referencia, y que en un primer momento podrían parecer innecesarios, son utilizados en el lugar oportuno. En este caso el concepto nombre_de_E-atributo se utiliza como parte de la expresión de la restricción. Observemos que en realidad el concepto que aparece explícito es el de nombre_de_atributo, pero gracias a la buena estructura del sistema de referencia la descripción del concepto nombre_de_E-atributo es obvia (puesto que un E-atributo es en particular un atributo, lo que a su vez se muestra muy explícitamente en el soporte parcial de atributos).

este estilo de notación en caso de ambigüedad.

En segundo lugar, a partir de esta restricción se puede deducir que el usuario/creador de la base de datos no especificará en ningún caso la creación de los E-atributos. La especificación de un atributo implica proporcionar un nombre y un dominio. En el caso de los E-atributos, el dominio está prefijado (es el E-dominio) y el nombre debe tomarse a partir del esquema de E-relación correspondiente, luego la definición de los E-atributos es siempre automática (como veremos, de hecho con los esquemas de E-relación sucede algo similar). Así pues, un ejemplo[9] de esquema de E-relación sería:

$$\langle empleado, \{empleadoç\}\rangle$$

esquema_de_P-relación ∗esquema_de_relación_n-ario	
con $n \geq 2$, tal que uno de sus atributos es E-atributo	
y el resto son P-atributos∗	[Cod79, p.413, sec 7] int.

Los esquemas de P-relación son una mejora de los esquemas de relacionales tradicionales del modelo relacional. La intención de uso de estos esquemas es que almacenen las 'propiedades' de las entidades, pero la introducción obligatoria en ellos del E-atributo (que actuará como clave principal oculta al usuario [Cod79, p.413, sec 7]) le aporta una importante potencia expresiva y operativa. Es necesario observar que nuestra descripción de esquema de P-relación es más restrictiva que la original de Codd, ya que él admite que un esquema de P-relación contenga otros E-atributos «cuyos roles serán puramente referenciales, es decir de clave foránea». Sin embargo ninguno de los ejemplos que plantea recogen esta situación, y entendemos que la existencia de otros tipos de esquemas (como veremos a continuación) permiten restringir la estructura de los esquemas de P-relación a la forma en que los hemos presentado. Un ejemplo de esquema de P-relación es:

$$\langle datos\_empleado, \{empleadoç, DNI, nombre, edad\}\rangle$$

Los esquemas de E-relación y P-relación son los únicos tipos de esquemas distinguidos que presenta Codd en el artículo de definición del RM/T. Sin embargo, compartimos el punto de vista de otros autores [RM83, EKTW87], según el cual, y teniendo en cuenta la intención de uso de diferentes tipos de entidad y en consecuencia de diferentes tipos de esquemas, es conveniente la introducción de los siguientes:

[9]Los ejemplos que utilizaremos en nuestra presentación son los utilizados por Codd [Cod79] o bien basados de forma directa en ellos.

esquema_de_C-relación ∗esquema_de_relación_binario tal que sus dos atributos son E-atributos∗	int.
esquema_de_A-relación ∗esquema_de_relación_n-ario, con $n \geq 3$, tal que todos sus atributos son E-atributos∗	int.

La utilización de las letras 'C' y 'A' en la nominación de estos conceptos está inspirada en el modo en el que Codd utiliza 'E' y 'P' en los esquemas de E-relación y P-relación respectivamente. Ya hemos observado que la 'E' se usa puesto que las E-relaciones se utilizarán para expresar la existencia de Entidades, y la 'P' puesto que las P-relaciones se usarán para expresar Propiedades de estas entidades. Anticipando parcialmente lo que vendrá más adelante, los esquemas de C-relación y de A-relación serán utilizados en la especificación de entidades **C***aracterísticas* y **A***sociativas*, respectivamente. Un ejemplo de esquema de C-relación es

$$<empleo\_empleado, \{empleoç, empleadoç\}>$$

y un ejemplo de esquema de A-relación es

$$<asign\_emp\_proy, \{asignaciónç, empleadoç, proyectoç\}>$$

esquema_de_C|A-relación ∗esquema_de_C-relación o esquema_de_A-relación∗	int.

El concepto esquema_de_C|A-relación se incluye por motivos exclusivamente técnicos, ya que su introducción nos permitirá enunciar algunas restricciones de forma más relajada.

esquema_de_N-relación ∗esquema_de_relación_n-ario, con $n \geq 2$, tal que al menos dos de sus atributos son E-atributos∗	int.

El concepto de esquema_de_N-relación no es introducido por Codd ni por ninguno de los autores antes citados. Aunque no lo presenta de manera explícita, Codd sí utiliza este tipo de esquemas para describir asociaciones-no-entidad [Cod79, p.417, sec 9.2] e introduce más adelante un valor de *tiporelación (reltype)*[10] para ellas [Cod79, p.425, sec 14]. Este valor es precisamente la letra 'N', por ser la inicial de 'no-entidad', y por ello la hemos elegido para nominar los esquemas de N-relación. Otros autores

[10]El *tiporelación* es un tipo de atributo definido de forma explícita en el modelo RM/T. Es un atributo que aparece una única vez en cada base de datos RM/T formando parte del *Catálogo* de la base de datos. Más adelante profundizaremos en la descripción del papel que juega este Catálogo.

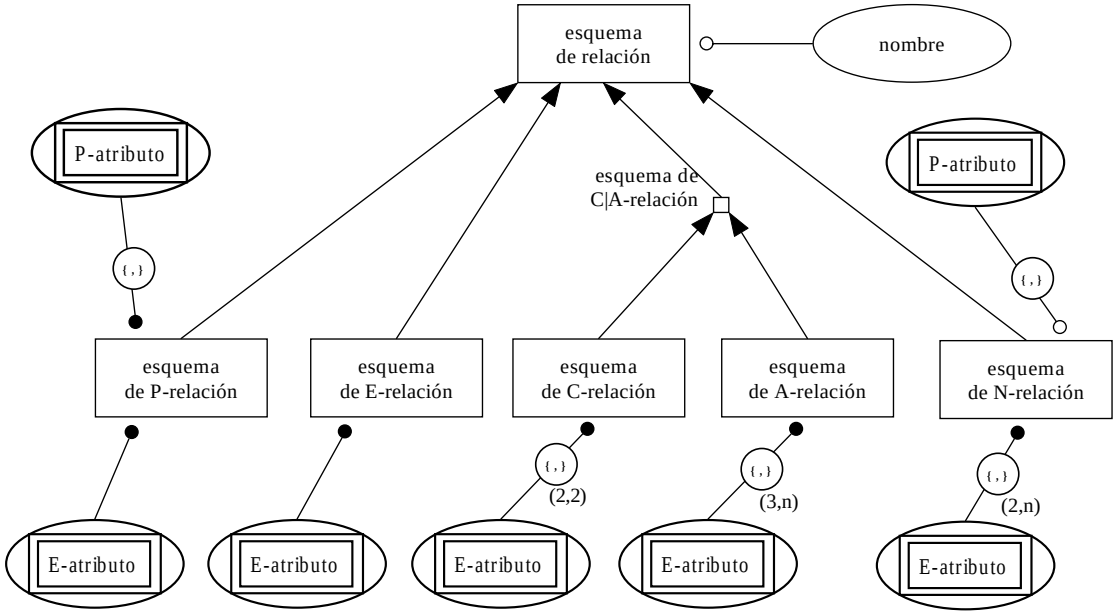


Figura 2.5: Soporte parcial. Esquemas de relación en RM/T

[Dat85, EW91] comentan la existencia de las asociaciones no-entidad en el modelo original, pero no las utilizan en la práctica. A nuestro juicio, una utilidad indirecta de las asociaciones no-entidad es que facilitarían la traducción de bases de datos relacionales clásicas hacia bases de datos RM/T, aunque sin duda este es un aspecto que necesita de un análisis en profundidad. Un ejemplo de esquema de N-relación es el siguiente:

$$<asignación, \{empleadoç, proyectoç, fecha\_de\_inicio\}>$$

Una vez definidos los tipos de esquemas de relación posibles, es conveniente incluir una restricción que imponga que éstos son los únicos tipos de esquemas permitidos en un esquema de base de datos RM/T.

Restricción G2	Todo esquema_de_relación es esquema_de_E-relación o esquema_de_P-relación o esquema_de_C-relación o esquema_de_A-relación o esquema_de_N-relación.

int.

Esta restricción está incluida implícitamente en el soporte parcial de esquemas de relación en el RM/T, que se muestra en la figura 2.5. Al igual que ocurría con el soporte parcial de atributos, a este soporte parcial, aun a pesar de no ser un

verdadero soporte, se le puede dar una interpretación en términos de sus conceptos fuente. En este caso, la interpretación coincide plenamente con la restricción que acabamos de imponer. Observemos además que este soporte parcial se apoya en el soporte parcial de atributos, pues utiliza los conceptos allí definidos por medio de atributos referenciales.

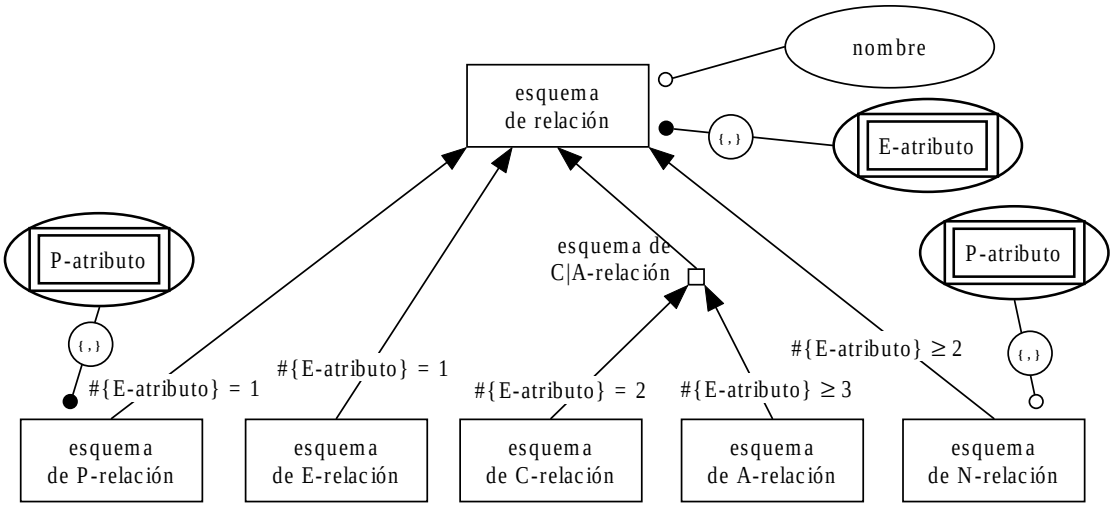


Figura 2.6: Soporte parcial. Esquemas de relación en RM/T. Notación alternativa

A la vista de las descripciones de los distintos tipos de esquemas de relación y más aún, a la vista del soporte parcial de la figura 2.5, es evidente que los diferentes tipos de esquemas tienen una descripción muy similar, o dicho de otro modo, que las atribuciones que es necesario especificar para describir los conceptos de los distintos tipos de esquemas son muy similares. Por ello en la figura 2.6 mostramos el mismo soporte parcial utilizando una notación diferente que entendemos mejora la legibilidad del soporte, describiendo el concepto sumidero esquema_de_relación mediante un conjunto de E-atributos y señalando una restricción para la cardinalidad de este conjunto para cada tipo de esquema de relación particular. Esta notación es una propuesta del presente trabajo, que creemos adecuada en este caso particular, pero que no forma parte de la notación estándar de la técnica NÓESIS.

Esquemas de molécula en RM/T

Hasta aquí la mayor parte (con las excepciones que hemos destacado) de los conceptos que hemos presentado pertenecen al modelo RM/T de forma explícita.

Los siguientes conceptos, por el contrario, no son introducidos de manera explícita por Codd ni por ninguno de los otros autores que han tratado el modelo RM/T. Sin embargo a nuestro juicio estos conceptos van a permitirnos presentar con mayor grado de precisión algunas de las ideas clave del modelo. Es conveniente volver a recordar aquí cual es uno de los objetivos de la creación de RM/T, que aparece en el abstract de [Cod79] y hemos citado con anterioridad:

> "La intención es la de capturar (de manera más o menos formal) más significado de los datos [...]. [...]La búsqueda de unidades con significado que sean mayores que la relación n-aria habitual – semántica molecular."

Sin embargo en el desarrollo del modelo Codd no presenta de manera totalmente explícita la existencia de *moléculas* en el RM/T, aunque la idea permanece implícitamente reflejada a lo largo de todo el texto de [Cod79].

esquema_de_molécula_sin_nombre ∗conjunto de esquemas_de_relación∗	int.
esquema_de_molécula_con_nombre ∗agregación de un nombre y un conjunto de esquemas_de_relación∗	int.
nombre_de_esquema_de_molécula_con_nombre ∗∗	int.
esquema_de_molécula ∗esquema_de_molécula_sin_nombre o esquema_de_molécula_con_nombre∗	int.

En un primer momento este concepto es bastante amplio, ya que *a priori* un esquema de molécula podría estar formado por cualquier conjunto de esquemas de relación. Como mostraremos de inmediato, en realidad no se admite la existencia de cualquier conjunto, sino de algunos con un significado preciso.

Por otra parte, hemos distinguido entre esquemas de molécula con y sin nombre por paralelismo con el concepto de esquema de relación. Sin embargo, la intención de uso de los esquemas de molécula, que está ligada a los tipos de entidad que se distinguen en RM/T, hace que con toda seguridad todos los esquemas de molécula que sean creados por un diseñador de bases de datos RM/T tengan nombre. Queda abierto el problema de analizar si es posible o no la creación *en tiempo de ejecución* de esquemas de molécula, de modo similar a como ocurre por ejemplo con la creación de esquemas de relación para la realización de *vistas* en el modelo relacional clásico. Esto esta íntimamente relacionado con los aspectos operacionales del modelo RM/T, que no trataremos aquí.

Como acabamos de comentar, Codd distinguen entre distintos tipos de entidad dentro de RM/T, y este punto constituye otro de los aspectos fundamentales del modelo. En concreto, Codd establece una clasificación de tipos de entidades percibidas,

> "[...]en función de si:
>
> 1. desempeñan un papel subordinado describiendo entidades de algún otro tipo, en cuyo caso se denominan *características*
> 2. desempeñan un papel superior [superordinado] interrelacionando entidades de otros tipos, en cuyo caso se denominan *asociativas*
> 3. no desempeñan ninguno de los papeles anteriores, en cuyo caso se denominan *núcleos*"

Esta clasificación es de tipo conceptual, de tipo semántico, ya que incide en el significado que para el diseñador tienen las entidades percibidas, así como los roles que desempeñan y las relaciones entre ellas. Sin embargo, y siendo uno de los puntos fuertes del modelo relacional su buena estructura formal, matemática, en el artículo de definición de RM/T no se definen estructuras explícitas para dar soporte a la clasificación de entidades propuesta. Sin embargo las ideas subyacentes a estas estructuras se encuentran implícitas en los ejemplos presentados y en la definición de ciertas componentes del Catálogo, así como en ciertas frases de Codd: «Es posible pensar en la colección de P-relaciones de una E-relación dada en términos de un *tipo de molécula de propiedad* [...]». En varias de entre el resto de referencias que tratan sobre RM/T [Dat92, EKTW87, RM83, EW91] también están presentes las ideas que nosotros presentamos de un modo más formal (probablemente [RM83] realiza el enfoque más parecido al nuestro).

esquema_de_molécula_núcleo ∗esquema_de_molécula con un esquema_de_E-relación y ningún, uno o varios esquemas_de_P-relación∗	int.
esquema_de_molécula_característica ∗esquema_de_molécula con un esquema_de_E-relación, un esquema_de_C-relación y ningún, uno o varios esquemas_de_P-relación∗	int.

esquema_de_molécula_asociativa ∗esquema_de_molécula con un esquema_de_E-relación, un esquema_de_A-relación y ningún, uno o varios esquemas_de_P-relación∗	int.
esquema_de_molécula_entidad ∗esquema_de_molécula_núcleo o esquema_de_molécula_característica o esquema_de_molécula_asociativa∗	int.

Destaquemos que a la vista de la descripción de los conceptos de esquemas de molécula entidad se admite la existencia de esquemas que no tengan ningún esquema de P-relación asociado.

Restricción L3	El nombre_de_esquema_de_relación del esquema_de_E-relación constituyente de un esquema_de_molécula_entidad es el mismo que el nombre_de_esquema_de_molécula.

int.

Lo que esta restricción impone es algo que ya habíamos anticipado previamente. El nombre de un esquemas de E-relación (y como consecuencia el nombre del E-atributo que pertenece al esquema) viene dado por el nombre del esquema de molécula (entidad) al que pertenece. Por tanto el diseñador de la base de datos RM/T especificará la creación de un esquema de molécula entidad (junto con su tipo –núcleo, característica o asociativa–) tras lo que el SGBD RM/T crearía automáticamente el esquema de E-relación (y los de C|A-relación, en su caso) correspondiente.

Restricción L4	El E-atributo de todo esquema_de_P-relación de un esquema_de_molécula_entidad es el mismo que el del esquema_de_E-relación del esquema_de_molécula.

[Cod79, p.413, sec 7], int.

Esta restricción es la base para que se pueda cumplir lo que Codd denomina *regla de integridad de propiedad* [Cod79, p.413, sec 7]. Esta restricción, junto con la que vendrá a continuación, es uno de los principios fundamentales en la definición del RM/T, ya que el E-atributo al que se hace referencia actuará como clave principal en todos los esquemas de relación de cada esquema de molécula. Observemos que tanto esta restricción como la siguiente es de tipo *intramolecular* ya que restringe la estructura de cada esquema de molécula particular.

esquema_de_molécula_entidad-no-núcleo ∗esquema_de_molécula_característica o esquema_de_molécula_asociativa∗	int.

Este concepto se incluye por razones técnicas, y de la misma forma que el concepto de esquema_de_C|A-relación, permite que la siguiente restricción se enuncie de forma mucho más sencilla.

Restricción L5	Uno de los E-atributos del esquema_de_C|A-relación de un esquema_de_molécula_entidad-no-núcleo es el mismo que el del esquema_de_E-relación del esquema_de_molécula.

int.

Esta es una restricción que no es introducida por Codd, ya que el no distingue C-relaciones ni A-relaciones, disponiendo exclusivamente de P-relaciones, con lo que con la restricción anterior a la última enunciada le basta. Sin embargo, otros autores, al introducir las C-relaciones y A-relaciones se ven obligados a introducir también, si bien en forma no explícita, esta restricción (por ejemplo [RM83, p. 148, sec 2.1]). La inclusión de la restricción nos lleva a los siguientes conceptos:

E-atributo_principal_de_esquema_de_molécula_entidad-no-núcleo ∗E-atributo que tienen en común todos los esquemas_de_relación de esquema_de_molécula_entidad-no-núcleo∗	int.

Observemos que no es necesario que esta definición incluya a los esquemas de molécula núcleo, ya que en ellos todos los esquemas de relación tienen un sólo E-atributo, que puesto que es único, es obviamente 'principal'.

E-atributo_no-principal_de_esquema_de_molécula_entidad-no-núcleo ∗cualquier E-atributo del esquema_de_C|A-relación de esquema_de_molécula_entidad-no-núcleo distinto del E-atributo principal∗	int.

Mostraremos a continuación distintos ejemplos de esquemas de moléculas entidad, que cumplen por tanto con las descripciones y restricciones impuestas hasta este momento. De nuevo los ejemplos proceden de, o están inspirados en, el artículo original de Codd.

Ejemplo de esquema de molécula núcleo:

$\langle empleado,$	[Nombre]
$\langle empleado, \{empleadoç\}\rangle,$	[Esq. de E-relación]
$\langle datos\_empleado, \{empleadoç, DNI, nombre, edad\}\rangle$	[Esq. de P-relación]
$\rangle$	

Ejemplo de esquema de molécula característica:

$\langle empleo,$	[Nombre]
$\langle empleo, \{empleoç\}\rangle,$	[Esq. de E-relación]
$\langle empleo\_empleado, \{empleoç, empleadoç\}\rangle,$	[Esq. de C-relación]
$\langle datos\_empleo, \{empleoç, tipo, fecha\}\rangle$	[Esq. de P-relación]
$\rangle$	

Ejemplo de esquema de molécula asociativa:

$\langle asignación,$	[Nombre]
$\langle asignación, \{asignaciónç\}\rangle,$	[Esq. de E-relación]
$\langle asign\_emp\_proy, \{asignaciónç, empleadoç, proyectoç\}\rangle,$	[Esq. de A-relación]
$\langle datos\_asignación, \{asignaciónç, fecha\}\rangle$	[Esq. de P-relación]
$\rangle$	

La familia de esquemas de molécula permitidas en RM/T se completa con el siguiente:

esquema_de_molécula_asociación-no-entidad ∗esquema_de_molécula que consta de un esquema_de_N-relación∗	int.

Como ya hemos observado, Codd no incluye esquemas de N-relación en su exposición, pero sí presenta el tipo asociación-no-entidad, para representar «objetos que interrelacionan entidades pero no tienen por sí mismos el status de entidad». También hemos comentado que, a nuestro juicio, los esquemas de N-relación pueden constituir una herramienta útil para realizar inmersiones de esquemas de bases de datos relacionales 'clásicas' en esquemas RM/T. Sin embargo la razón que aduce Codd para incluir las asociaciones-no-entidad es algo más oscura [Cod79, p.412, sec 6]:

"La principal razón para incluir asociaciones-no-entidad en RM/T es de

tipo expositivo: mostrar lo débiles que son estas asociaciones en contraste con las entidades asociativas."

En particular observemos que un esquema de molécula asociación-no-entidad no contiene ningún esquema de E-relación, y por tanto no es posible hacer referencia a este tipo de esquemas de molécula desde cualquiera de los otros tipos. En particular las asociaciones-no-entidad no pueden tener características, ni ser característica de otras entidades, ni formar parte de entidades asociativas.

Restricción L6	El nombre_de_esquema_de_molécula_asociación-no-entidad es el mismo que el del esquema_de_N-relación del esquema_de_molécula.

int.

El siguiente ejemplo de esquema de molécula asociación-no-entidad (versión alternativa del ejemplo de esquema de molécula asociativa) es también utilizado por Codd.

$\langle asignación,$ [Nombre]
$\quad \langle asignación, \{empleadoç, proyectoç, fecha\} \rangle$ [Esq. de N-relación]
$\rangle$

De modo análogo a lo que sucedía con los esquemas de relación, la inclusión de la siguiente restricción es un refuerzo de la interpretación del soporte parcial de esquemas de molécula que mostramos en la figura 2.7.

Restricción G3	Todo esquema_de_molécula es esquema_de_molécula_entidad o esquema_de_molécula_asociación-no-entidad.

int.

Los aspectos más básicos de la parte intensional del modelo RM/T se completan con el siguiente concepto y algunas restricciones.

esquema_de_base_de_datos_RM/T ∗conjunto de esquemas_de_molécula∗	int.

Siendo rigurosos, este concepto debe pertenecer con mayor propiedad a la componente de definición de modelo del metamodelo Nóesis, ya que es una parte fundamental para la definición de lo que es una base de datos RM/T (concepto que

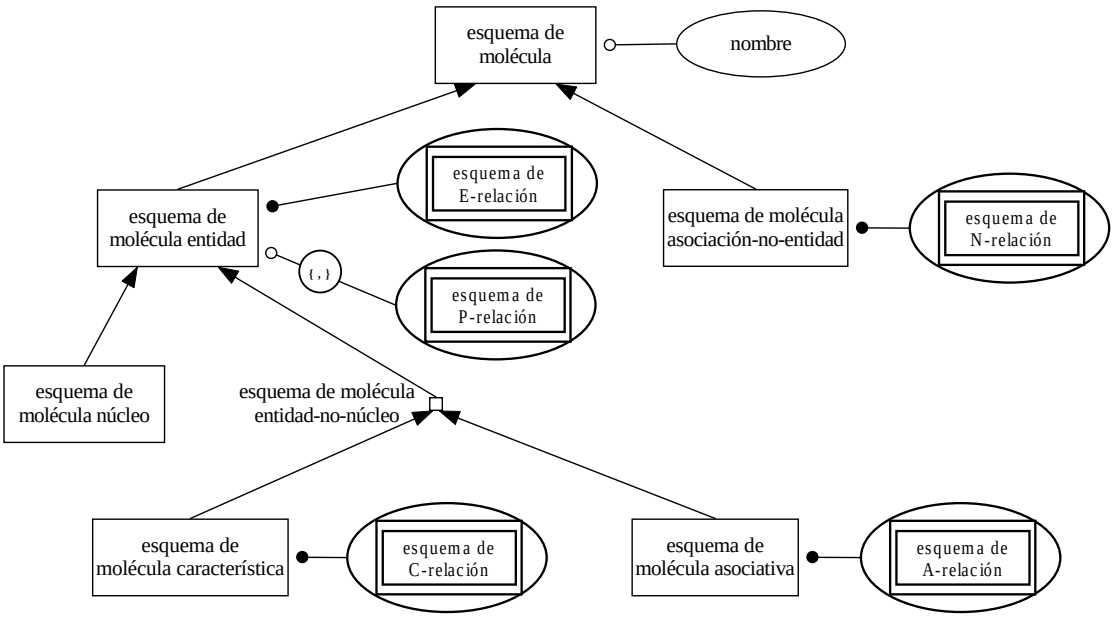


Figura 2.7: Soporte parcial. Esquemas de molécula en RM/T

es precisamente el que queremos que sea un modelo del metamodelo que estamos construyendo). Se incluye aquí fundamentalmente para aclarar el significado de las restricciones que enunciaremos a continuación.

Observemos que de hecho un esquema de base datos RM/T está compuesto exclusivamente por un conjunto de esquemas de molécula entidad (los cuales pueden ser a su vez esquemas de molécula núcleo, característica o asociativa) y un conjunto de esquemas de molécula asociación-no-entidad. La descripción del concepto puede abreviarse debido a la inclusión de los distintos conceptos y restricciones que hemos ido mostrando. Además deben incluirse varias restricciones de carácter global.

Restricción G4	Todos los nombres_de_esquema_de_relación de un esquema_de_base_de_datos_RM/T deben ser distintos. En particular, y como consecuencia, todos los nombres_de_esquema_de_molécula deben ser distintos.

int.

Restricción G5	Todo esquema_de_base_de_datos_RM/T contiene al menos un esquema_de_molécula_núcleo.

int.

La mínima expresión de un esquema de base de datos RM/T es un único esquema de molécula núcleo. A partir de ahí la inclusión de nuevos esquemas de molécula puede conllevar la inclusión obligatoria de otros. Por ejemplo, si un esquema de base de datos RM/T contiene un esquema de molécula asociativa binario (entendiendo éste por aquel que contiene dos E-atributos no principales), el esquema de base de datos debe contener al menos otros dos esquemas de molécula entidad. De modo análogo, si un esquema de base de datos RM/T contiene un esquema de molécula asociación-no-entidad binario (entendiendo éste por aquel tal que su esquema de N-relación contiene dos E-atributos), también el esquema de base de datos debe contener al menos dos esquemas de molécula entidad. Estas situaciones son determinadas por las dos siguientes restricciones.

Restricción G6	Para cada E-atributo_no-principal_de_esquema_de_molécula_entidad-no-núcleo, debe existir en el esquema_de_base_de_datos_RM/T un esquema_de_molécula_entidad cuyo esquema_de_E-relación lo tenga como E-atributo.

[Cod79, p.415, sec 8], int.

[Cod79, p.416, sec 9.1], int.

Esta última restricción recoge la parte intensional de dos reglas introducidas explícitamente por Codd, la *regla de integridad de característica* y la *regla de integridad de asociación*. Es importante notar que todas las reglas que Codd introduce centran su atención en el aspecto extensional y no en el intensional, en el que nosotros nos encontramos. Esto hace que en el caso de la regla de integridad de asociación, Codd admita como posible que se desconozca alguna de las entidades participantes en una asociación. Observemos sin embargo que *lo que potencialmente puede desconocerse es el surrogate correspondiente*, pero evidentemente el esquema de molécula asociativa debe recoger que cuáles son las entidades que se están asociando, luego siempre deben conocerse los E-atributos correspondientes. Véase que esta restricción así como la siguiente son restricciones de carácter *intermolecular*, ya que afectan simultáneamente a dos esquemas de molécula diferentes.

Restricción G7	Para cada E-atributo del esquema_de_N-relación de un esquema_de_molécula_asociación-no-entidad, debe existir en el esquema_de_base_de_datos_RM/T un esquema_de_molécula_entidad cuyo esquema_de_E-relación lo tenga como E-atributo.

int.

Aunque este principio no es introducido explícitamente por Codd (ya que Codd ni tan siquiera recoge el concepto esquema_de_ N-relación) su necesidad es clara a la vista de la intención con la que se incluyen los esquemas de molécula asociación-no-entidad.

Como conclusión de la parte intensional básica, mostramos en la figura 2.8 el soporte parcial correspondiente a los conceptos incluidos en esta parte. El hecho de que sea tan sólo un soporte parcial está en relación con una observación previa. Este soporte define un aspecto parcial de lo que es una base de datos RM/T, exactamente el aspecto relativo a lo que es la intensión de una tal base de datos. En cierto sentido podríamos hablar de un 'modelo' de dicho soporte, que, según la idea general de la componente de definición de modelo de todo metamodelo NÓESIS, estaría compuesto por instancias de cada uno de los conceptos fuente de este soporte. En este caso concreto este supuesto 'modelo' estaría formado por un conjunto de esquemas de molécula núcleo, un conjunto de esquemas de molécula característica, un conjunto de esquemas de molécula asociativa y un conjunto de esquemas de molécula asociación-no-entidad, lo que coincide exactamente con la descripción del concepto esquema_de_base_de_datos_RM/T que hemos mostrado. Sin embargo hay que insistir en que el objetivo final del metamodelo es el de definir que es una base de datos RM/T, de la cual el esquema es sólo una parte, la parte intensional. Por otra parte observemos que este soporte parcial es de alguna forma la unión de los soportes parciales de atributos (figura 2.4), esquemas de relación (figura 2.5) y esquemas de molécula (figura 2.7), y proporciona la necesaria visión de conjunto que no se obtiene con los otros soportes parciales.

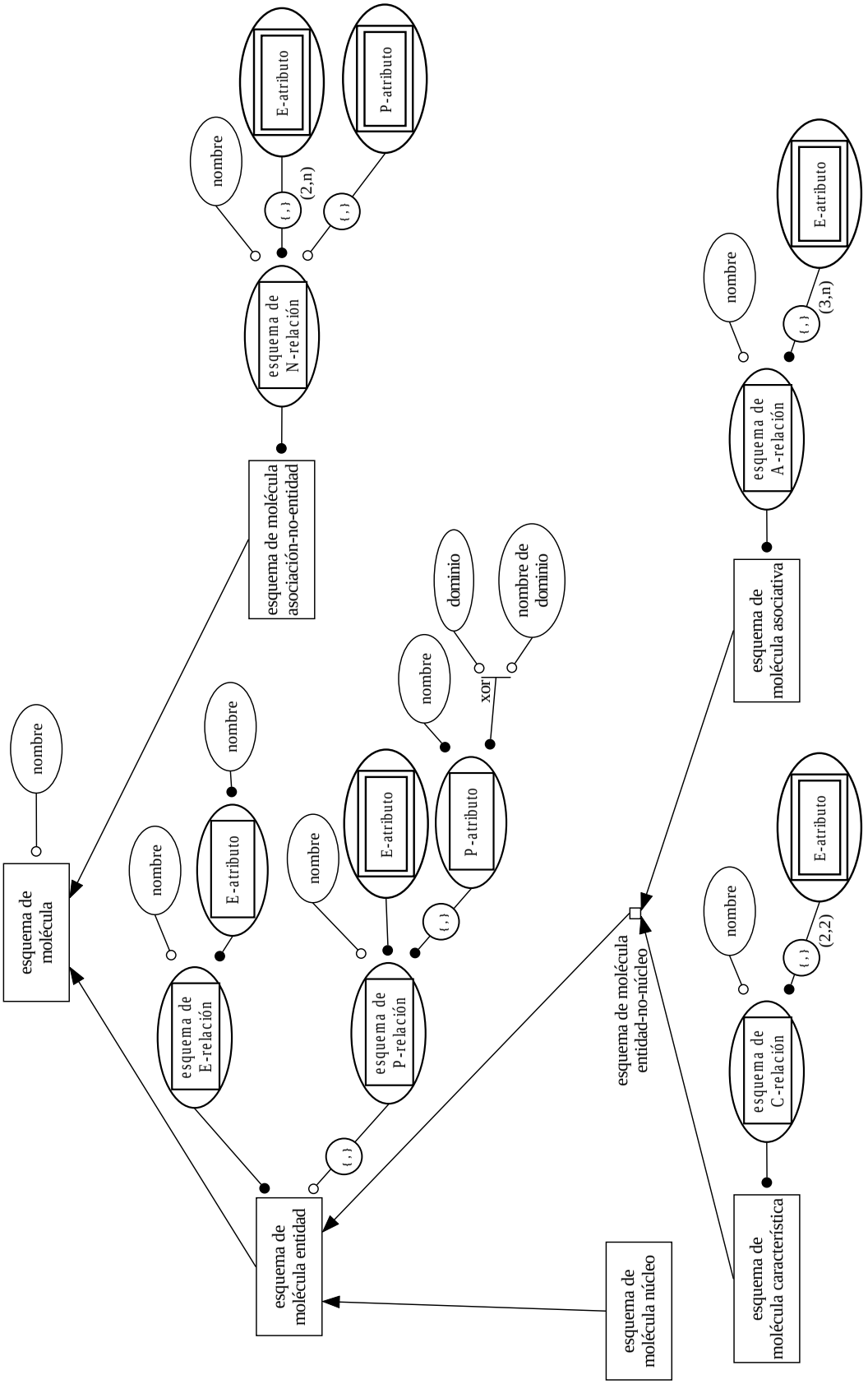


Figura 2.8: Soporte parcial. Parte intensional de bases de datos RM/T

2.2.3. Un metamodelo Nóesis de RM/T: parte extensional básica

Una vez concluida la revisión de los elementos fundamentales del aspecto intensional de RM/T, vamos a concentrarnos en el aspecto extensional. Es sobre este aspecto sobre el que más explícitamente trata el artículo de definición de RM/T. Con nuestro enfoque, la parte extensional va ligada a la parte intensional, de forma que en muchos casos los 'conceptos extensionales' se apoyan en 'conceptos intensionales' que han sido descritos en la sección anterior.

Fundamentos de la parte extensional del modelo RM/T

valor_de_dominio ∗∗	[Cod79, p.399, sec 2.1], int.

Codd describe un *dominio simple* como aquel en el que todos sus valores son atómicos. La aparición de dominios que no sean simples implicaría el incumplimiento de la Primera Forma Normal del modelo relacional básico. Sin embargo el metamodelo que estamos presentando se ocupa exclusivamente de los aspectos estructurales fundamentales, y por tanto no aborda los problemas derivados de la normalización de relaciones. En concreto, recordemos que según nuestra descripción, un dominio es simplemente un conjunto de valores sin ninguna restricción adicional. Observemos además que la noción de *valor atómico* (definida por Codd como aquel que es «indescomponible por el sistema gestor de bases de datos») dista mucho de estar clara o aceptada comúnmente.

valor_de_atributo ∗agregación del nombre_de_atributo y de un valor_de_dominio_de_atributo∗	int.

Observemos que el concepto valor_de_dominio_de_atributo no ha aparecido descrito explícitamente. Sin embargo su significado es obvio a partir de los conceptos dominio_de_atributo y valor_de_dominio que sí han sido introducidos. Observemos que esta descripción de valor_de_atributo incluye tanto el valor en sí como el nombre del atributo correspondiente. Aunque sin utilizar el nombre valor_de_atributo, este es el enfoque usado por Codd en [Cod79] (si bien recordemos que para Codd un atributo es en realidad sólo un índice, un nombre). Por tanto, ejemplos de valores

de atributo para los atributos que a su vez hemos mostrado como ejemplo en la sub-sección anterior son:

$\langle persona, juan\rangle$ o $\langle persona, xyz\rangle$ para el atributo $\langle persona, varchar(40)\rangle$

$\langle color, rojo\rangle$ para el atributo $\langle color, \{rojo, verde, azul\}\rangle$

$\langle edad, 25\rangle$ para el atributo $\langle edad, \{n; 0 \leq n \leq 120\}\rangle$

tupla_de_esquema_de_relación ∗conjunto de valores_de_atributo, uno y sólo uno por cada atributo de esquema_de_relación∗	[Cod79, p.399, sec 2.1], int.

Sin duda este es uno de los conceptos fundamentales no sólo del modelo RM/T sino también del modelo relacional básico. Cualquier esquema de relación (que recordemos puede tener nombre o no) consta de un conjunto de atributos. Tomando un valor de atributo para cada uno de los atributos de este conjunto se forma un nuevo conjunto, denominado tupla del esquema de relación. Un ejemplo de tupla es:

$\{\langle nombre, Juan\rangle, \langle edad, 25\rangle, \langle DNI, 12345678\rangle\}$

para el esquema de relación $\langle empleado, \{DNI, nombre, edad\}\rangle$

Observemos que en este ejemplo hemos mostrado en distinto orden la tupla y el conjunto de atributos del esquema de relación. Hemos hecho esto, intencionadamente, para destacar que estos conjuntos son, en efecto, conjuntos, y por tanto *no tienen por qué estar ordenados*. Esta propiedad es básica en el modelo relacional, que se define casi estrictamente de forma conjuntista, de tal forma que el orden en el que se especifican los atributos del conjunto que forma parte de cualquier esquema de relación es indistinto. De ahí que sea importante que la definición de valor de atributo incluya no sólo el valor del dominio correspondiente sino también el nombre del atributo. Como es bien conocido (e incluso a veces confundido con los fundamentos del modelo) una forma diferente de expresar sintácticamente tuplas es mediante una representación tabular:

DNI	nombre	edad
12345678	Juan	25

Insistamos en que esto es sólo una notación, una sintaxis concreta para la noción de tupla de esquema de relación.

relación_de_esquema_de_relación_sin_nombre ∗conjunto de tuplas_de_esquema_de_relación_sin_nombre∗	int.
relación_de_esquema_de_relación_con_nombre ∗agregación del nombre_de_esquema_de_relación_con_nombre y un conjunto de tuplas_de esquema_de_relación_con_nombre∗	int.
relación_de_esquema_de_relación ∗relación_de_esquema_de_relación_sin_nombre o relación_de_esquema_de_relación_con_nombre∗	int.

Sin lugar a dudas estos son los conceptos fundamentales del modelo relacional (ya que, sin ir más lejos, dan nombre al propio modelo), y por ello creemos necesario hacer algunas observaciones. En primer lugar, nos hemos visto obligados a introducir distintos conceptos de relación en función de si su esquema de relación asociado tiene nombre o no, puesto que ese ha sido nuestro enfoque en la parte intensional del modelo.

En segundo lugar, creemos conveniente resaltar que en estos conceptos se recoge una idea que puede parecer 'obvia' pero que entendemos que es fundamental que se muestre de forma explícita para que la definición del modelo sea correcta: en una relación, para que pueda ser llamada tal, todas sus tuplas deben ser tuplas del *mismo* esquema de relación. En nuestro enfoque esta propiedad se consigue gracias a que nuestra descripción de tupla recoge explícitamente el hecho de que una tupla es tupla de un esquema de relación concreto. Otra posible definición de tupla (más parecida a la usada por Codd) sería considerarla como un conjunto de valores de atributo sin más, sin que estos atributos estuvieran ligados a ningún esquema de relación. Observemos que sin embargo Codd, al dar su definición de relación, dice que «una relación consiste en un conjunto de tuplas, teniendo cada tupla el mismo conjunto de atributos», con lo que implícitamente se esta refiriendo al concepto de esquema de relación. Por otra parte, observemos también que aunque es sencillo comprender el significado de la definición de Codd, su formulación es bastante imprecisa, ya que hace referencia al "conjunto de atributos de una tupla", noción no definida formalmente, y más a la vista de que según Codd, «una tupla [es] un conjunto de pares (A:v), donde A es un atributo y v es un valor obtenido del dominio de A».

En tercer lugar, observemos que las descripciones que hemos proporcionado no admiten la existencia de tuplas repetidas en relaciones. Esta propiedad se deduce simplemente del hecho de que cuando nos referimos a la noción matemática de 'con-

junto' para decir que una relación contiene un conjunto de tuplas, estamos haciendo referencia a la idea de conjunto (estructura matemática) más habitual, en la que no se admiten repetidos. Habitualmente un conjunto en el que se admiten repetidos es llamado *conjunto con repetición* o *multiconjunto*. Codd ha defendido siempre la necesidad de la prohibición de tuplas repetidas en las relaciones, llegando incluso a afirmar que un sistema gestor de bases de datos (SGBD) que permita la existencia de tuplas repetidas no puede ser llamado con propiedad SGBD relacional [Cod90, p.5–6, p.17–19]. Sin embargo esta idea no es compartida por otros autores, y no se aplica en muchos SGBD comerciales en la actualidad.

Por último, hemos de admitir que la nominación de los conceptos relativos a la noción de 'relación' es compleja, pero no debe perderse de vista que nuestra intención es que el metamodelo (y en particular el sistema de referencia del mismo) rompa cualquier ambigüedad posible. Sin duda cualquier meta-analista o cualquier analista familiarizado mínimamente con el modelo relacional encontrará redundantes estos conceptos (o al menos sus nombres), ya que es bastante claro que detrás de una relación subyace un esquema de relación o que cualquier esquema de relación se define con la intención de crear una relación basada en él. De hecho, en el resto de nuestra exposición usaremos frecuentemente el término *relación* para referir al concepto relación_de_esquema_de_relación.

Un ejemplo de relación de esquema de relación sin nombre es:

$$\{\{\langle nombre, Juan\rangle, \langle edad, 25\rangle, \langle DNI, 12345678\rangle\},$$
$$\{\langle edad, 34\rangle, \langle DNI, 87654321\rangle, \langle nombre, María\rangle\}\}$$
para el esquema de relación (sin nombre) $\{DNI, nombre, edad\}$

En este ejemplo volvemos a remarcar el hecho de que los valores de atributo dentro de una tupla no tienen porque estar ordenados, y por ello la relación presentada como ejemplo tiene dos tuplas de aspecto aparentemente distinto. En el siguiente ejemplo, de relación de esquema de relación con nombre, repetimos la representación tabular que hemos mostrado con anterioridad.

empleado

DNI	nombre	edad
12345678	Juan	25
87654321	María	34

Observemos que la sintaxis tabular implica una 'reordenación' de los valores de atributo de las tuplas, para que la representación tenga en efecto la forma de tabla, pero debemos insistir en que esta reordenación se produce a un nivel exclusivamente sintáctico. En esta sintaxis el nombre de la relación se muestra justo encima de la tabla.

Valores de atributos en RM/T

Al igual que ocurría en la parte intensional, los conceptos que hemos visto hasta ahora pertenecen en realidad al modelo relacional clásico. El primer concepto extensional RM/T "puro" es:

surrogate ∗valor_de_E-dominio∗	[Cod79, p.410, sec 4], int.

Recordemos las ideas fundamentales que están tras la noción de surrogate, la mayoría de las cuales ya han sido comentadas brevemente en la parte intensional. Un surrogate es un valor, asignado por el sistema, que siempre estará oculto al usuario de la base de datos. Un usuario no puede crear, borrar o modificar (al menos no de manera explícita) los surrogates. La intención de uso de un surrogate es la de identificar de manera única y permanente a cada entidad percibida, lo que en particular implica que todos los surrogates identificadores de entidades en una base de datos RM/T serán distintos, o dicho de otro modo, que dos surrogates serán iguales en una base de datos si y sólo si denotan la misma entidad en el mundo percibido de entidades. Remarquemos también que un surrogate es un valor de E-dominio y no de [E]-dominio. Aún a riesgo de parecer reiterativos, hay que recordar que *el* [E]-dominio puede interpretarse como el conjunto de todos los E-dominios, que en particular el [E]-dominio no es dominio, y que por tanto no es posible hablar de sus valores.

valor_de_E-atributo ∗agregación del nombre_de_E-atributo y un surrogate∗	[Cod79, p.410, sec 4], int.

valor_de_P-atributo ∗agregación del nombre_de_P-atributo y un valor_de_dominio_de_P-atributo∗	int.

Observemos que podría parecer que estos conceptos son redundantes. Puesto

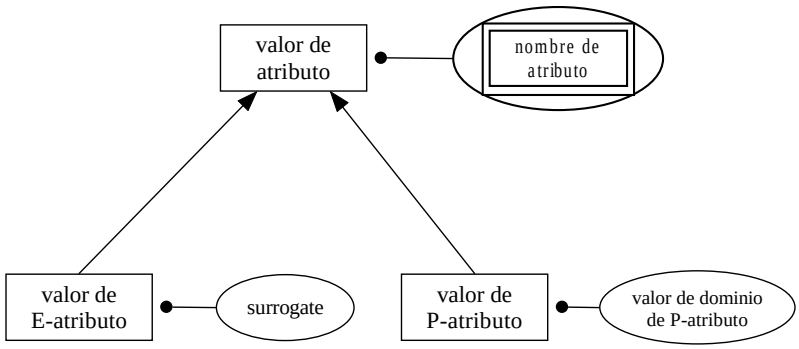


Figura 2.9: Soporte parcial. Valores de atributos en RM/T

que hemos mostrado la descripción del concepto de valor_de_atributo, y según la parte intensional, los E-atributos y los P-atributos no son sino especializaciones de la noción de atributo, la descripción de los conceptos valor_de_E-atributo y valor_de_P-atributo es poco menos que automática. Sin embargo, hay dos razones para mostrar su descripción explícita. Por una parte hemos pretendido que quedara constancia explícita del hecho de que los valores de E-atributo contienen surrogates y no valores de dominios cualesquiera (lo cual sin duda es obvio a la vista de la descripción –intensional– de E-atributo). Por otra hemos de tener en cuenta que en una base de datos RM/T *los únicos valores de atributo posible son los de E-atributo y P-atributo* (ya que estos son los únicos atributos que se pueden definir), y que en realidad el concepto valor_de_atributo sólo se utiliza de forma auxiliar para comprender mejor la descripción de los conceptos relativos a valores de atributos que acabamos de exponer.

Puesto que la parte extensional del modelo se apoya, se basa en la parte intensional, para cada uno de los soportes parciales que se han mostrado en la parte intensional tendremos un correspondiente en la parte extensional, soportes parciales que también incluimos en el metamodelo. Así por ejemplo la figura 2.9 es el soporte parcial 'extensional' correspondiente al soporte parcial 'intensional' de la figura 2.4 que hemos mostrado en la página 80.

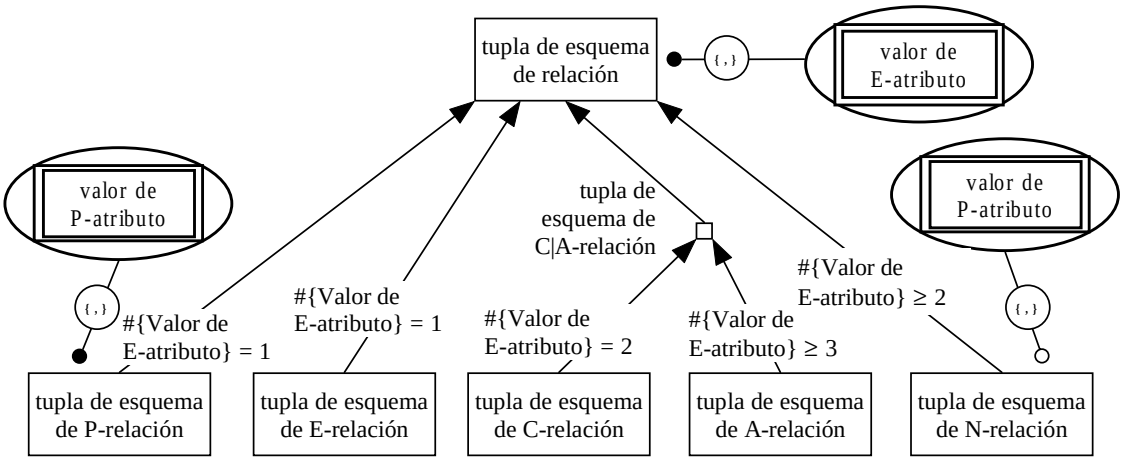


Figura 2.10: Soporte parcial. Tuplas en RM/T

Relaciones en RM/T

tupla_de_esquema_de_E-relación ∗conjunto formado por un único valor_de_E-atributo∗	[Cod79, p.410, sec 5], int.
tupla_de_esquema_de_P-relación ∗conjunto formado un valor_de_E-atributo y un valor_de_P-atributo por cada P-atributo del esquema_de_P-relación∗	[Cod79, p.413, sec 7], int.
tupla_de_esquema_de_C-relación ∗conjunto formado por un valor_de_E-atributo por cada E-atributo del esquema_de_C-relación∗	int.
tupla_de_esquema_de_A-relación ∗conjunto formado por un valor_de_E-atributo por cada E-atributo del esquema_de_A-relación∗	int.
tupla_de_esquema_de_N-relación ∗conjunto formado por un valor_de_E-atributo por cada E-atributo del esquema_de_N-relación y un valor_de_P-atributo por cada P-atributo del esquema_de_N-relación ∗	int.

Tal y como ocurre con el concepto valor_de_atributo, el concepto tupla_de_esquema_de_relación se incluye como concepto auxiliar para unificar y facilitar la expresión de los tipos de tuplas particulares que se admiten en RM/T, las

cuales están, lógicamente, ligados uno a uno con los tipos de esquemas de relación que se permiten en el modelo. Aunque la descripción de estos tipos de tupla es poco menos que evidente a la vista de la descripción general de tupla y de los distintos tipos de esquemas de relación en RM/T, incluimos por completitud estos conceptos, así como su soporte parcial asociado en la figura 2.10 (correspondiente con el soporte parcial de esquemas de relación de la figura 2.5 o de la figura 2.6 –con notación alternativa, más parecida a la usada en la figura 2.10– en páginas 84 y 85).

Otro ejemplo de la 'deducción' de descripciones de conceptos lo encontramos al intentar describir los distintos tipos de relaciones posibles en RM/T. Como hemos observado con anterioridad, el concepto de relación es sin duda el concepto fundamental del modelo relacional. Del mismo modo, la distinción entre tipos de relaciones es uno de los puntos básicos del modelo RM/T. Como muestra de la potencia expresiva de la técnica NÓESIS, en este caso vamos a expresar de dos modos distintos las descripciones de los tipos de relación. En la descripción textual, es decir, en el sistema de referencia, las descripciones se basan en el hecho de que todas las relaciones en RM/T son en particular relaciones, sólo que basadas en tipos distintos de esquemas de relación. Por su parte, en la descripción gráfica, es decir, en el soporte parcial, las descripciones se basan en que las relaciones en RM/T, puesto que son relaciones, están formadas por un conjunto de tuplas, aunque para cada tipo de relación se utiliza el tipo de tupla apropiado en base a las descripciones de tipos de tupla anteriores. Observemos que el soporte parcial de relaciones en la figura 2.11, no se corresponde de manera exacta con ninguno de los soportes parciales de la parte intensional si bien está ligado al soporte parcial de esquemas de relación de la figura 2.5.

E-relación ∗relación de esquema_de_E-relación∗	[Cod79, p.410, sec 5], int.
P-relación ∗relación de esquema_de_P-relación∗	[Cod79, p.413, sec 7], int.
C-relación ∗relación de esquema_de_C-relación∗	int.
A-relación ∗relación de esquema_de_A-relación∗	int.
C|A-relación ∗C-relación o A-relación∗	int.
N-relación ∗relación de esquema_de_N-relación∗	int.

Ya hemos observado que en cualquier relación no se permite la aparición de tuplas repetidas. Este hecho tiene ciertas interpretaciones particulares que es interesante destacar para los distintos tipos de relaciones presentados. En el caso de E-relación,

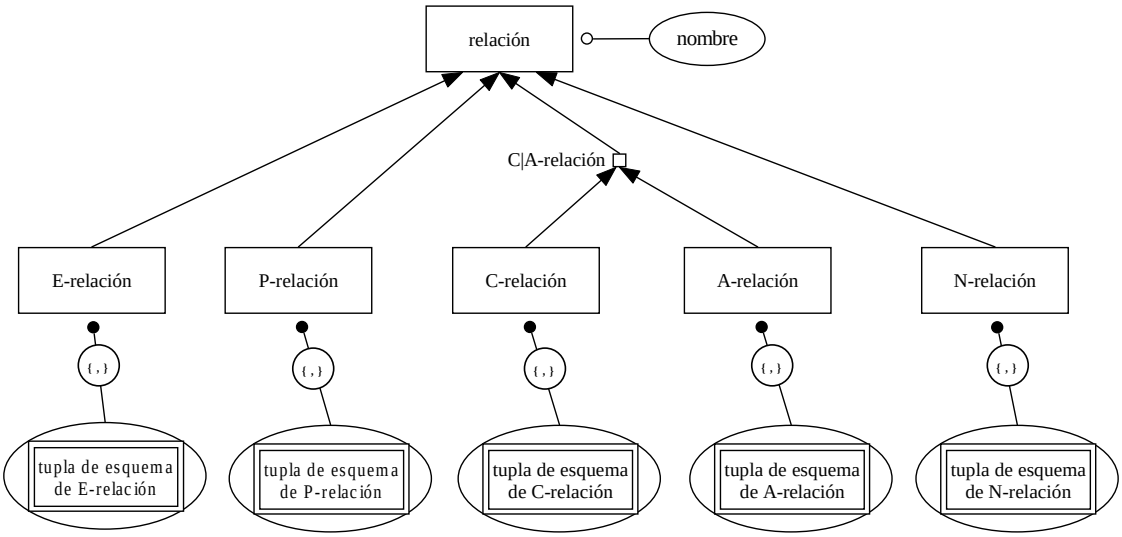


Figura 2.11: Soporte parcial. Relaciones en RM/T

cada una de las tuplas contenidas en una E-relación consta sólo de un valor de atributo. En particular cada uno de estos valores es un valor de E-atributo formado a su vez por el nombre del E-atributo de la E-relación y un surrogate. Por tanto, en este caso particular podemos afirmar de modo informal que lo que una E-relación almacena es un conjunto de surrogates. Así es como precisamente lo presenta Codd en [Cod79, p.411, sec 5]: «El principal propósito de una E-relación es listar todos los surrogates de las entidades que tienen ese tipo y que están almacenados en la base de datos en un momento concreto». En este caso, por tanto, el hecho de que una relación no admita tuplas repetidas implica que en una E-relación no existen surrogates repetidos. Esto implementa en modo simbólico, la propiedad ya repetida en varias ocasiones de que si dos surrogates son iguales en una base de datos RM/T es porque están designando la misma entidad. El principio de no-repetición de tuplas impide que una misma entidad se almacene de forma duplicada en una E-relación.

Sin embargo, en el caso de P-relaciones, es necesario imponer una restricción adicional a la prohibición de existencia de tuplas repetidas para evitar que se produzcan algunas situaciones indeseadas. En el siguiente ejemplo de P-relación no hay tuplas repetidas, y sin embargo hay un valor de E-atributo repetido en dos tuplas distintas:

datos_empleado

empleadoç	DNI	nombre	edad
α	12345678	Juan	25
β	87654321	María	34
α	13579246	José	30

En este ejemplo por primera vez hemos necesitado, por razones expositivas, utilizar una notación concreta para los surrogates. Recordemos que en una base de datos RM/T los surrogates han de estar ocultos al usuario. Sin embargo, en la explicación textual de los ejemplos se hace imprescindible la utilización de una cierta notación para que sean comprensibles. De hecho, la sintaxis concreta que vamos a utilizar (letras griegas para cada surrogate) es exactamente la misma que utiliza Codd en [Cod79].

La razón por la que una P-relación como esta no es válida es determinada por la intención de uso de las P-relaciones. Citamos a Codd: «Cada P-relación tiene como clave principal un E-atributo». Por tanto los valores de E-atributo de cada tupla de una P-relación (recordemos que una P-relación sólo tiene un E-atributo) deben ser distintos. Esta importante restricción ha de ser incluida en nuestro metamodelo:

Restricción L7	Dos tuplas_de_P-relación deben tener diferentes valores_de_E-atributo.

int.

Un primer comentario respecto a esta restricción es que en ella hacemos referencia al concepto tupla_de_P-relación, que no se encuentra explícitamente definido. Sin embargo, la existencia explícita de los conceptos tupla_de_esquema_de_relación y relación_de_esquema_de_relación hacen que el significado de la noción de tupla_de_relación (y todos sus asociados, tales como tupla_de_P-relación) sea obvia. De hecho, los distintos autores que han tratado el modelo relacional en la literatura hacen mención con más frecuencia a 'tupla de relación' que a 'tupla de esquema de relación', aunque bajo nuestro punto de vista la descripción del modelo es más correcta con nuestro enfoque. Observemos como curiosidad que hasta este momento, en que hemos tenido que imponer restricciones de tipo extensional, no ha sido necesario realizar esta aclaración sobre el término 'tupla'.

Además, también será necesario incluir una restricción análoga para las C-

relaciones y A-relaciones. Recordemos que en realidad Codd no habla explícitamente ni de C-relaciones ni de A-relaciones, aunque usa como ejemplos ciertas relaciones cuya estructura corresponde con éstas (recordemos también que otros autores, como [RM83] sí que introducen estas relaciones al tratar sobre RM/T). En realidad Codd trata nuestras C|A-relaciones como si fueran P-relaciones, sin más. Curiosamente, para definir de forma correcta la restricción de 'clave principal' Codd se ve obligado a introducir, si bien de manera muy informal, el concepto de esquema de molécula (y consecuentemente molécula) que nosotros ya hemos introducido. De nuevo en palabras de Codd: «Para cada tipo de entidad e y cada par de P-relaciones para e, los únicos atributos que tienen en común estas relaciones son sus claves principales. El rol de este E-atributo es el de identificador único para la relación en la que aparece [...]. Por tanto, cada P-relación tiene exactamente un E-atributo que tiene [este rol]». Este rol se corresponde con lo que nosotros hemos llamado E-atributo_principal_de_esquema_de_molécula_entidad-no-núcleo. Ya hemos observado que no tiene mucho sentido hablar de E-atributo principal para esquemas de molécula núcleo, puesto que todos los esquemas de relación de este tipo de esquemas contienen un único E-atributo. Por lo tanto, para dar una restricción análoga a la anterior pero aplicada al caso de las C|A-relaciones, es más adecuado introducir primero los conceptos relativos a la noción de molécula.

Moléculas en RM/T

molécula_de_esquema_de_molécula ∗agregación del nombre_de_esquema_de_molécula y un conjunto de relaciones, una por cada esquema_de_relación del esquema_de_molécula∗	int.

Esta descripción es imprecisa del mismo modo que lo es la de esquema_de_molécula, puesto que sólo serán admitidos algunos tipos de moléculas concretos (al igual que en el caso de relaciones, usaremos simplemente el nombre molécula cuando queramos referir al concepto molécula_de_esquema_de_molécula).

molécula_núcleo ∗molécula de esquema_de_molécula_núcleo∗ ∗molécula que consta de una E-relación y ninguna, una o varias P-relaciones∗	int.
molécula_característica ∗molécula de esquema_de_molécula_característica∗ ∗molécula que consta de una E-relación, una C-relación y ninguna, una o varias P-relaciones∗	int.
molécula_asociativa ∗molécula de esquema_de_molécula_asociativa∗ ∗molécula que consta de una E-relación, una A-relación y ninguna, una o varias P-relaciones∗	int.
molécula_asociación-no-entidad ∗molécula de esquema_de_molécula_asociación-no-entidad∗ ∗molécula que consta de una N-relación∗	int.

Los conceptos anteriores muestran una característica de la técnica NÓESIS y en particular del sistema de referencia que no habíamos tenido necesidad de utilizar hasta ahora. Para cualquier concepto del sistema de referencia se admite la posibilidad de incluir más de una descripción. La razón fundamental que puede motivar esta múltiple (habitualmente doble) descripción es la de aclarar el significado del concepto en cuestión. Así ocurre en el caso de los distintos tipos de moléculas: puesto que entendemos que la noción de molécula es fundamental dentro del metamodelo, es adecuado aclarar su significado en función de la estructura de relaciones que contiene cada uno de los tipos. Destaquemos que sin embargo las primeras descripciones, derivadas de los diferentes esquemas de molécula, son más precisas que las segundas, puesto que en efecto una molécula ha de estar basada en un esquema de molécula, lo que implica que se han de cumplir todas las restricciones que para esta estructura se han impuesto. La descripción del tipo 'nombre unido a un conjunto de relaciones' es más débil en este sentido.

Ya estamos en condiciones de formular la restricción que necesitábamos imponer para las C|A-relaciones.

Restricción L8	Dos tuplas_de_C|A-relación deben tener diferentes valores_de_E-atributo_principal.

int.

Recordemos que el E-atributo principal de esquema de molécula entidad-no-núcleo es el único E-atributo que tienen en común todos los esquemas de relación del esquema de molécula, y de este modo coincide con la cita de Codd a la que hacíamos referencia anteriormente: el E-atributo principal desempeña el papel de clave principal en la C|A-relación de la molécula entidad-no-núcleo, concepto que describimos a continuación.

molécula_entidad ∗molécula_núcleo o molécula_característica o molécula_asociativa∗	int.
molécula_entidad-no-núcleo ∗molécula_característica o molécula_asociativa∗	int.

Destacamos las moléculas de tipo entidad –núcleo, característica y asociativa– puesto que estos tipos son los auténticamente relevantes para RM/T (recordemos que las asociaciones-no-entidad desempeñan un papel secundario), y las moléculas de tipo entidad que no son núcleo para enunciar cómodamente las siguientes restricciones.

Restricción L9	El valor_de_E-atributo de toda tupla_de_P-relación de molécula_entidad debe ser valor_de_E-atributo de una tupla_de_E-relación de la molécula.

[Cod79, p.413, sec 7], int.

La anterior restricción se corresponde con la parte extensional de la denominada por Codd *regla de integridad de propiedad* [Cod79, p.413, sec 7]. Puesto que, como ya hemos reiterado en diversas ocasiones, nuestro planteamiento incluye C|A-relaciones que no son incluidas por Codd, el cumplimiento de la *regla de integridad de propiedad* necesita de la inclusión de una restricción adicional.

Restricción L10	El valor_de_E-atributo_principal de toda tupla_de_C|A-relación de molécula_entidad-no-núcleo debe ser valor_de_E-atributo de una tupla_de_E-relación de la molécula.

[Cod79, p.413, sec 7], int.

Expliquemos en términos menos formales lo que las dos últimas restricciones imponen. La primera de ellas dice que una propiedad (apareciendo en una P-relación)

debe ser propiedad de una entidad existente (apareciendo en una E-relación). La segunda, que para especificar a quién describe o quienes asocia una entidad característica o asociativa respectivamente (lo que aparece en una C-relación o A-relación), dicha entidad característica o asociativa debe, en efecto, existir (aparecer en la E-relación). En el modelo relacional clásico este tipo de restricciones serían restricciones de clave foránea, pero las estructuras especiales existentes en RM/T hacen que sea posible dotar de más significado a las restricciones.

Observemos también que las restricciones en principio no dicen nada en sentido contrario. Es decir, puede existir una entidad (lo cual se reflejaría con la existencia de un surrogate en la E-relación correspondiente) *sin que se especificaran propiedades para ella* (sin tuplas correspondientes en las P-relaciones adecuadas). Y de forma análoga hipotéticamente podría existir una entidad no-núcleo (característica o asociativa) sin que se especificaran lo caracterizado o los asociados (es decir, sin que apareciera el correspondiente surrogate como valor de E-atributo principal en C|A-relación). Esta última situación podría significar que para una entidad no-núcleo conociésemos sus propiedades (tuplas de P-relación) pero no a qué entidad caracteriza o a qué entidades asocia. Sin embargo esta situación no se permite para entidades no-núcleo, lo que se deduce, de manera indirecta, a partir dos reglas enunciadas por Codd: la *regla de integridad de característica* [Cod79, p.415, sec 8] y la *regla de integridad de asociación* [Cod79, p.416, sec 9.1]. La existencia de estas reglas hacen necesaria la introducción de restricciones adicionales, de las cuales la primera es:

Restricción L11	El valor_de_E-atributo de toda tupla_de_E-relación de molécula_entidad-no-núcleo debe ser valor_de_E-atributo_principal de una tupla_de_C|A-relación de la molécula.

[Cod79, p.413, sec 7], int.

[Cod79, p.415, sec 8], int.

[Cod79, p.416, sec 9.1], int.

Esta última restricción junto con las dos anteriores son restricciones de carácter *intramolecular*, ya que afectan a cada molécula particular. Del mismo modo que en la parte intensional hemos enunciado restricciones de carácter intramolecular que limitan la estructura de los esquemas de molécula, en este caso las restricciones actúan a nivel extensional, limitando las posibles tuplas que pueden aparecer en

las relaciones que forman una molécula. Vamos a ilustrar el significado de estas restricciones mediante un ejemplo.

coche (molécula característica)

coche (E-relación)

cocheç
α
γ
β

coche_propietario (C-relación)

cocheç	personaç
α	ϵ
β	δ
γ	ϵ

datos_coche (P-relación)

cocheç	marca	matrícula
β	Seat	1234 BLG
α	Fiat	5678 BDF

Este ejemplo muestra una molécula característica de nombre *coche* compuesta de una E-relación, una C-relación (ambas obligatorias en la estructura de cualquier molécula característica) y una P-relación. El nombre de la E-relación es también *coche* lo que está impuesto por la restricción *L3* (en página 88), el nombre de la C-relación es *coche_propietario* y el nombre de la P-relación es *datos_coche*. Observemos como las tres restricciones anteriores se cumplen, ya que: (1) todos los valores de E-atributo en la P-relación aparecen en la E-relación (la inversa no se cumple: el surrogate γ aparece en la E-relación pero no en la P-relación, lo que significa que aunque el coche γ existe en la base de datos, no conocemos sus propiedades); (2) todos los valores de E-atributo principal en la C-relación aparecen en la E-relación (el E-atributo principal de la molécula es el único que tienen en común todas las relaciones, en este caso *cocheç*); y (3) todos los valores de E-atributo en la E-relación aparecen en el E-atributo principal de la C-relación. Observemos también que ninguna de las restricciones anteriores imponen nada especial sobre los E-atributos no principales de una molécula entidad-no-núcleo. En el caso particular del ejemplo, las restricciones no dicen nada sobre el E-atributo *personaç*.

De modo análogo a lo que sucedía en la parte intensional, la siguiente restricción refuerza la interpretación del soporte parcial de moléculas que mostramos en la figura 2.12.

Restricción G8	Toda molécula es molécula_entidad o molécula_asociación-no-entidad.

int.

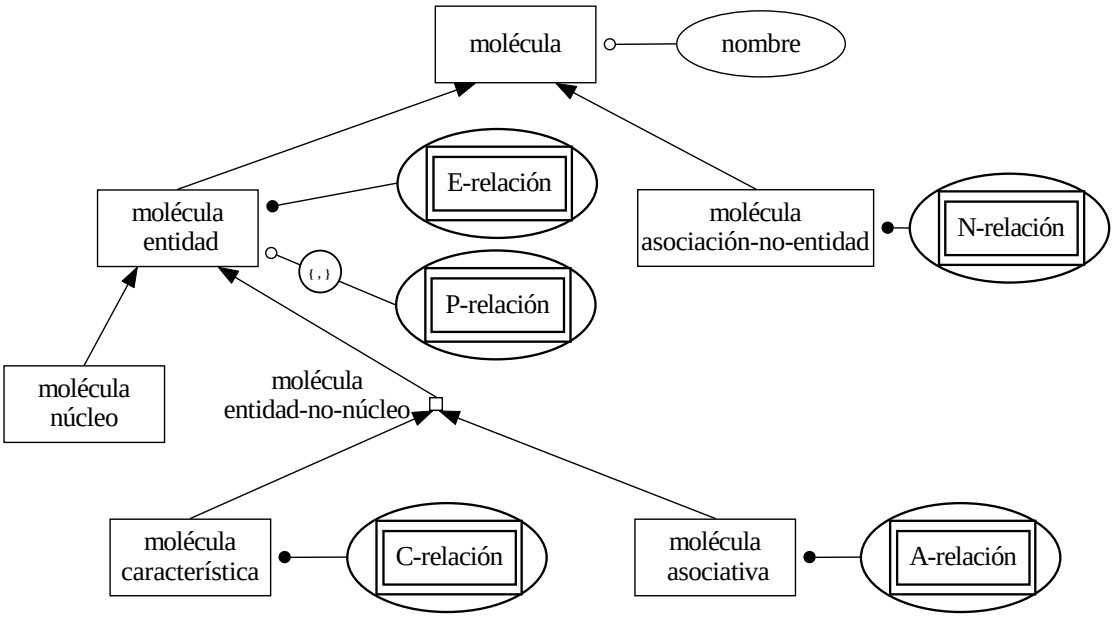


Figura 2.12: Soporte parcial. Moléculas en RM/T

El último concepto que enunciaremos constituye en realidad la definición de modelo RM/T.

base_de_datos_RM/T ∗conjunto de moléculas∗	int.

Ya hemos comentado en la sub-sección correspondiente de la parte intensional que la descripción de esquema_de_base_de_datos_RM/T considerada independientemente era insuficiente. Del mismo modo, una base_de_datos_RM/T es en realidad un conjunto de moléculas entidad (con al menos una molécula núcleo, y ninguna, una o varias moléculas entidad-no-núcleo) y un conjunto (eventualmente vacío) de moléculas asociación-no-entidad, cumpliéndose además todas las restricciones impuestas hasta ahora y también las dos siguientes de carácter *intermolecular*.

Restricción G9	Cada valor_de_E-atributo_no-principal de toda tupla_de_C|A-relación de molécula_entidad-no-núcleo debe ser valor_de_E-atributo de una tupla_de_E-relación en una molécula_entidad de la base_de_datos_RM/T.

[Cod79, p.415, sec 8], int.

[Cod79, p.416, sec 9.1], int.

En términos informales, lo que esta restricción exige es que en una entidad característica debe 'conocer' a su caracterizado y una entidad asociativa debe 'conocer' a sus asociados. Esto es así ya que en una tupla de C|A-relación, debido a su especial estructura, disponemos de un valor de E-atributo principal y uno o varios (en función de si la relación es C- o A-relación) valores de E-atributo no-principal. El valor de E-atributo principal en la tupla (que por las restricciones anteriores debe ser también valor de E-atributo en la E-relación de la molécula) representa a la entidad característica o asociativa, y el o los valores de E-atributo no-principal representan las entidades caracterizadas o asociadas.

La restricción recoge de manera parcial las ya nombradas *regla de integridad de característica* [Cod79, p.415, sec 8] y *regla de integridad de asociación* [Cod79, p.416, sec 9.1]. Sin embargo la restricción no se corresponde idénticamente con estas reglas, puesto que aunque Codd obliga a que «una entidad característica no puede existir en la base de datos a menos que la entidad que describe más inmediatamente también esté en la base de datos», y por tanto la restricción enunciada se ajusta a lo impuesto para características, por el contrario afirma que «una entidad asociativa puede existir incluso aunque una o más de las entidades participantes en esa asociación sean desconocidas. En tal caso, el surrogate *E-nulo* se usa para indicar que una entidad participante es desconocida». Nuestra opinión, compartida por Date en su revisión de RM/T [Dat92, p.295], es que permitir valores nulos en valores de E-atributo es un planteamiento que necesita un estudio muy detallado acerca de sus ventajas e inconvenientes[11], por lo que en la restricción anterior no admitimos esta situación.

Para ilustrar el significado de la noción de base de datos RM/T, y en particular de los últimos conceptos y restricciones enunciados veamos el siguiente ejemplo.

persona (molécula núcleo)

persona *(E-relación)*

personaç
ϵ
δ

nombre_persona *(P-relación)*

personaç	nombre
δ	José
ϵ	Juan

DNI_persona *(P-relación)*

personaç	DNI
δ	1234567

[11] El estudio de la utilidad de (y los problemas por) el uso de valores nulos es tema recurrente en la investigación sobre bases de datos. A modo de ejemplo citaremos [KCL87] y [HR96] ya que en ambas referencias se trata el caso de RM/T.

coche (molécula característica)

coche *(E-relación)*

cocheç
α
γ
β

coche_propietario *(P-relación)*

cocheç	personaç
α	ϵ
β	δ
γ	ϵ

datos_coche *(P-relación)*

cocheç	marca	matrícula
β	Seat	1234 BLG
α	Fiat	5678 BDF

Este pequeño ejemplo de base de datos RM/T consta de dos moléculas, una de tipo núcleo (de nombre *persona*) y otra de tipo característica (la molécula *coche* que ya habíamos mostrado anterioridad), en la que se comprueba con facilidad que todas las restricciones impuestas se verifican. En particular observemos el cumplimiento de la última restricción, ya que los dos valores distintos de E-atributo no-principal en la C-relación de la molécula *coche* aparecen como valores de E-atributo en la E-relación de la molécula *persona*.

La última restricción del metamodelo afecta a las moléculas asociación-no-entidad.

Restricción G10	Cada valor_de_E-atributo de toda tupla_de_N-relación de molécula_asociación-no-entidad debe ser valor_de_E-atributo de una tupla_de_E-relación en una molécula_entidad de la base_de_datos_RM/T.

[Cod79, p.417, sec 9.2], int.

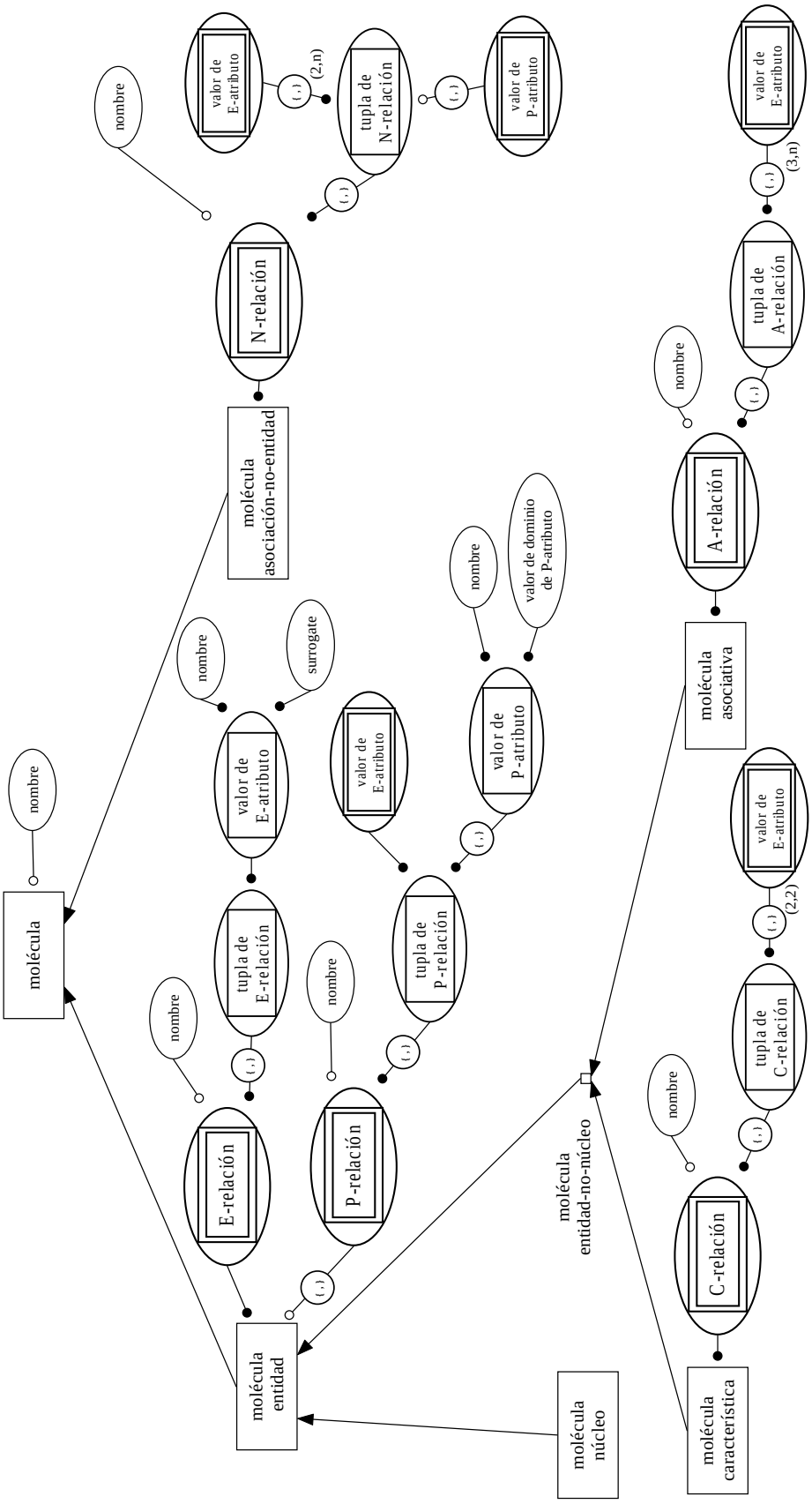


Figura 2.13: Soporte. Base de datos RM/T

La figura 2.13 constituye el soporte principal del metamodelo NÓESIS de RM/T, ya que una base de datos RM/T se obtiene como modelo de este soporte. El metamodelo NÓESIS de RM/T se completa con la perspectiva (tabla 2.3, página 73) el sistema de referencia, que mostramos de forma agrupada en la tabla 2.4, el conjunto de restricciones locales en la tabla 2.5 y el conjunto de restricciones globales en la tabla 2.6.

Tabla 2.4: Sistema de referencia para RM/T

dominio ∗conjunto de valores de tipo similar∗	[Cod79, p.399, sec 2.1]
nombre_de_dominio ∗∗	[Cod79, p.424, sec 14] int.
atributo ∗agregación de un nombre y un dominio o nombre_de_dominio∗	[Cod79, p.399, sec 2.1] int.
nombre_de_atributo ∗∗	[Cod79, p.424, sec 14] int.
dominio_de_atributo ∗∗	int.
esquema_de_relación_sin_nombre ∗conjunto de atributos∗	[Cod79, p.414, sec 7] int.
esquema_de_relación_con_nombre ∗agregación de un nombre y un conjunto de atributos∗	[Cod79, p.414, sec 7] int.
nombre_de_esquema_de_relación_con_nombre ∗∗	int.
esquema_de_relación ∗esquema_de_relación_sin_nombre o esquema_de_relación_con_nombre∗	[Cod79, p.399, sec 2.1] int.
dominio_de_esquema_de_relación ∗producto cartesiano de los dominios_de_atributo de esquema_de_relación∗	int.
grado_de_esquema_de_relación ∗número de atributos de esquema_de_relación∗	[Cod79, p.399, sec 2.1] int.
esquema_de_relación_n-ario ∗esquema_de_relación de grado n∗	[Cod79, p.408, sec 3] int.
[E]-dominio ∗dominio que se elige entre un conjunto de dominios y se nomina E-dominio∗	[Cod79, p.410, sec 4] int.
E-atributo ∗atributo con dominio E-dominio∗	[Cod79, p.410, sec 4] int.
P-atributo ∗atributo que no es E-atributo∗	int.
esquema_de_E-relación ∗esquema_de_relación_unario cuyo atributo es E-atributo∗	[Cod79, p.410, sec 5] int.

continúa en página siguiente

viene de página anterior	
esquema_de_P-relación ∗esquema_de_relación_n-ario con $n \geq 2$, tal que uno de sus atributos es E-atributo y el resto son P-atributos∗	[Cod79, p.413, sec 7] int.
esquema_de_C-relación ∗esquema_de_relación_binario tal que sus dos atributos son E-atributos∗	int.
esquema_de_A-relación ∗esquema_de_relación_n-ario, con $n \geq 3$, tal que todos sus atributos son E-atributos∗	int.
esquema_de_C|A-relación ∗esquema_de_C-relación o esquema_de_A-relación∗	int.
esquema_de_N-relación ∗esquema_de_relación_n-ario, con $n \geq 2$, tal que al menos dos de sus atributos son E-atributos∗	int.
esquema_de_molécula_sin_nombre ∗conjunto de esquemas_de_relación∗	int.
esquema_de_molécula_con_nombre ∗agregación de un nombre y un conjunto de esquemas_de_relación∗	int.
nombre_de_esquema_de_molécula_con_nombre ∗∗	int.
esquema_de_molécula ∗esquema_de_molécula_sin_nombre o esquema_de_molécula_con_nombre∗	int.
esquema_de_molécula_núcleo ∗esquema_de_molécula con un esquema_de_E-relación y ningún, uno o varios esquemas_de_P-relación∗	int.
esquema_de_molécula_característica ∗esquema_de_molécula con un esquema_de_E-relación, un esquema_de_C-relación y ningún, uno o varios esquemas_de_P-relación∗	int.
esquema_de_molécula_asociativa ∗esquema_de_molécula con un esquema_de_E-relación, un esquema_de_A-relación y ningún, uno o varios esquemas_de_P-relación∗	int.
continúa en página siguiente	

viene de página anterior

esquema_de_molécula_entidad ∗esquema_de_molécula_núcleo o esquema_de_molécula_característica o esquema_de_molécula_asociativa∗		int.
esquema_de_molécula_entidad-no-núcleo ∗esquema_de_molécula_característica o esquema_de_molécula_asociativa∗		int.
E-atributo_principal_de_esquema_de_molécula_entidad-no-núcleo ∗E-atributo que tienen en común todos los esquemas_de_relación de esquema_de_molécula_entidad-no-núcleo∗		int.
E-atributo_no-principal_de_esquema_de_molécula_entidad-no-núcleo ∗cualquier E-atributo del esquema_de_C|A-relación de esquema_de_molécula_entidad-no-núcleo distinto del E-atributo principal∗		int.
esquema_de_molécula_asociación-no-entidad ∗esquema_de_molécula que consta de un esquema_de_N-relación∗		int.
esquema_de_base_de_datos_RM/T ∗conjunto de esquemas_de_molécula∗		int.
valor_de_dominio ∗∗	[Cod79, p.399, sec 2.1],	int.
valor_de_atributo ∗agregación del nombre_de_atributo y de un valor_de_dominio_de_atributo∗		int.
tupla_de_esquema_de_relación ∗conjunto de valores_de_atributo, uno y sólo uno por cada atributo de esquema_de_relación∗	[Cod79, p.399, sec 2.1],	int.
relación_de_esquema_de_relación_sin_nombre ∗conjunto de tuplas_de_esquema_de_relación_sin_nombre∗		int.
relación_de_esquema_de_relación_con_nombre ∗agregación del nombre_de_esquema_de_relación_con_nombre y un conjunto de tuplas_de esquema_de_relación_con_nombre∗		int.

continúa en página siguiente

viene de página anterior	
relación_de_esquema_de_relación ∗relación_de_esquema_de_relación_sin_nombre o relación_de_esquema_de_relación_con_nombre∗	int.
surrogate ∗valor_de_E-dominio∗	[Cod79, p.410, sec 4], int.
valor_de_E-atributo ∗agregación del nombre_de_E-atributo y un surrogate∗	[Cod79, p.410, sec 4], int.
valor_de_P-atributo ∗agregación del nombre_de_P-atributo y un valor_de_dominio_de_P-atributo∗	int.
tupla_de_esquema_de_E-relación ∗conjunto formado por un único valor_de_E-atributo∗	[Cod79, p.410, sec 5], int.
tupla_de_esquema_de_P-relación ∗conjunto formado un valor_de_E-atributo y un valor_de_P-atributo por cada P-atributo del esquema_de_P-relación∗	[Cod79, p.413, sec 7], int.
tupla_de_esquema_de_C-relación ∗conjunto formado por un valor_de_E-atributo por cada E-atributo del esquema_de_C-relación∗	int.
tupla_de_esquema_de_A-relación ∗conjunto formado por un valor_de_E-atributo por cada E-atributo del esquema_de_A-relación∗	int.
tupla_de_esquema_de_N-relación ∗conjunto formado por un valor_de_E-atributo por cada E-atributo del esquema_de_N-relación y un valor_de_P-atributo por cada P-atributo del esquema_de_N-relación ∗	int.
E-relación ∗relación de esquema_de_E-relación∗	[Cod79, p.410, sec 5], int.
P-relación ∗relación de esquema_de_P-relación∗	[Cod79, p.413, sec 7], int.
C-relación ∗relación de esquema_de_C-relación∗	int.
A-relación ∗relación de esquema_de_A-relación∗	int.
C|A-relación ∗C-relación o A-relación∗	int.
N-relación ∗relación de esquema_de_N-relación∗	int.
	continúa en página siguiente

viene de página anterior	
molécula_de_esquema_de_molécula ∗agregación del nombre_de_esquema_de_molécula y un conjunto de relaciones, una por cada esquema_de_relación del esquema_de_molécula∗	int.
molécula_núcleo ∗molécula de esquema_de_molécula_núcleo∗ ∗molécula que consta de una E-relación y ninguna, una o varias P-relaciones∗	int.
molécula_característica ∗molécula de esquema_de_molécula_característica∗ ∗molécula que consta de una E-relación, una C-relación y ninguna, una o varias P-relaciones∗	int.
molécula_asociativa ∗molécula de esquema_de_molécula_asociativa∗ ∗molécula que consta de una E-relación, una A-relación y ninguna, una o varias P-relaciones∗	int.
molécula_asociación-no-entidad ∗molécula de esquema_de_molécula_asociación-no-entidad∗ ∗molécula que consta de una N-relación∗	int.
molécula_entidad ∗molécula_núcleo o molécula_característica o molécula_asociativa∗	int.
molécula_entidad-no-núcleo ∗molécula_característica o molécula_asociativa∗	int.
base_de_datos_RM/T ∗conjunto de moléculas∗	int.

Tabla 2.5: Restricciones locales para RM/T

[L1] Todos los atributos de un esquema_de_relación deben tener distinto nombre_de_atributo.	[Cod79, p.399, sec 2.1], int.
continúa en página siguiente	

viene de página anterior	
[L2] El nombre_de_E-atributo de esquema_de_E-relación es el mismo nombre que el del esquema añadiendo al final el carácter 'ç'.	[Cod79, p.410, sec 5], int.
[L3] El nombre_de_esquema_de_relación del esquema_de_E-relación constituyente de un esquema_de_molécula_entidad es el mismo que el nombre_de_esquema_de_molécula.	int.
[L4] El E-atributo de todo esquema_de_P-relación de un esquema_de_molécula_entidad es el mismo que el del esquema_de_E-relación del esquema_de_molécula.	[Cod79, p.413, sec 7], int.
[L5] Uno de los E-atributos del esquema_de_C|A-relación de un esquema_de_molécula_entidad-no-núcleo es el mismo que el del esquema_de_E-relación del esquema_de_molécula.	int.
[L6] El nombre_de_esquema_de_molécula_asociación-no-entidad es el mismo que el del esquema_de_N-relación del esquema_de_molécula.	int.
[L7] Dos tuplas_de_P-relación deben tener diferentes valores_de_E-atributo.	int.
[L8] Dos tuplas_de_C|A-relación deben tener diferentes valores_de_E-atributo_principal.	int.
[L9] El valor_de_E-atributo de toda tupla_de_P-relación de molécula_entidad debe ser valor_de_E-atributo de una tupla_de_E-relación de la molécula.	[Cod79, p.413, sec 7], int.
[L10] El valor_de_E-atributo_principal de toda tupla_de_C|A-relación de molécula_entidad-no-núcleo debe ser valor_de_E-atributo de una tupla_de_E-relación de la molécula.	[Cod79, p.413, sec 7], int.
	continúa en página siguiente

viene de página anterior	
[L11] El valor_de_E-atributo de toda tupla_de_E-relación de molécula_entidad-no-núcleo debe ser valor_de_E-atributo_principal de una tupla_de_C|A-relación de la molécula.	[Cod79, p.413-5-6, sec 7-8-9.1], int.

Tabla 2.6: Restricciones globales para RM/T

[G1] El [E]-dominio no es dominio	int.
[G2] Todo esquema_de_relación es esquema_de_E-relación o esquema_de_P-relación o esquema_de_C-relación o esquema_de_A-relación o esquema_de_N-relación.	int.
[G3] Todo esquema_de_molécula es esquema_de_molécula_entidad o esquema_de_molécula_asociación-no-entidad.	int.
[G4] Todos los nombres_de_esquema_de_relación de un esquema_de_base_de_datos_RM/T deben ser distintos. En particular, y como consecuencia, todos los nombres_de_esquema_de_molécula deben ser distintos.	int.
[G5] Todo esquema_de_base_de_datos_RM/T contiene al menos un esquema_de_molécula_núcleo.	int.
[G6] Para cada E-atributo_no-principal_de_esquema_de_molécula_entidad-no-núcleo, debe existir en el esquema_de_base_de_datos_RM/T un esquema_de_molécula_entidad cuyo esquema_de_E-relación lo tenga como E-atributo.	[Cod79, p.415-6, sec 8-9.1], int.

continúa en página siguiente

viene de página anterior	
[G7] Para cada E-atributo del esquema_de_N-relación de un esquema_de_molécula_asociación-no-entidad, debe existir en el esquema_de_base_de_datos_RM/T un esquema_de_molécula_entidad cuyo esquema_de_E-relación lo tenga como E-atributo.	int.
[G8] Toda molécula es molécula_entidad o molécula_asociación-no-entidad.	int.
[G9] Cada valor_de_E-atributo_no-principal de toda tupla_de_C|A-relación de molécula_entidad-no-núcleo debe ser valor_de_E-atributo de una tupla_de_E-relación en una molécula_entidad de la base_de_datos_RM/T.	[Cod79, p.415-6, sec 8-9.1], int.
[G10] Cada valor_de_E-atributo de toda tupla_de_N-relación de molécula_asociación-no-entidad debe ser valor_de_E-atributo de una tupla_de_E-relación en una molécula_entidad de la base_de_datos_RM/T.	[Cod79, p.417, sec 9.2], int.

2.2.4. Aspectos avanzados de RM/T

En las secciones anteriores hemos desarrollado con un gran nivel de detalle un metamodelo NÓESIS de parte del modelo RM/T. En efecto, el metamodelo que hemos presentado se concentra los aspectos más básicos, de tipo estructural del modelo. Sin embargo, RM/T incluye otra serie de características que es necesario analizar. Algunas de estas características podrían ser abordadas desde una perspectiva de metamodelización, aunque otras, fundamentalmente las que tratan los aspectos relacionados con diseño de bases de datos e intenciones de uso del modelo no son sencillas de abordar desde esta perspectiva. En las siguientes sub-secciones vamos a tratar algunas de estas cuestiones si bien con un planteamiento simplemente expositivo.

Operadores y catálogo

En nuestro metamodelo no hemos tratado el aspecto operacional del modelo RM/T, centrándonos de lleno en el aspecto estructural y parcialmente en los aspectos de integridad (restricciones). Este planteamiento es calificado por el propio Codd como incompleto, proporcionando una acertada analogía: «[La] estructura sin los correspondientes operadores o técnicas de inferencia es algo así como anatomía sin fisiología» [Cod79, p.398]. En concreto, RM/T incluye todos los operadores del álgebra relacional clásica (unión, selección, proyección, etc.) y además una serie de operadores específicos del RM/T, entre los cuales un conjunto importante lo constituyen los *operadores grafo*. Estos operadores se incluyen en el modelo para la manipulación de ciertas relaciones especiales llamadas *relaciones de grafo directas*. Estas relaciones desempeñan un papel curioso en la definición de RM/T, ya que, para cada base de datos RM/T existe una única relación de grafo por cada tipo de relación de grafo posible. Por ejemplo, cada base de datos RM/T incluye: una única *PG-relación* (PG: Property Graph) una relación binaria que liga los nombres de cada E-relación con sus P-relaciones asociadas; o una única *CG-relación* (CG: Characteristic Graph) una relación que liga el nombre de cada E-relación característica con el de la E-relación caracterizada (algo análogo sucede con la *AG-relación*). Observemos que estos aspectos estarán en realidad ocultos al usuario de la base de datos, y por tanto son aspectos del modelo que deben ser tenidos en cuenta principalmente por desarrolladores de sistemas de gestión de bases de datos RM/T. Ésta es la razón fundamental por la que no las hemos incluido de forma explícita en nuestro metamodelo. Tengamos en cuenta sin embargo que en cierta medida la necesidad de tales relaciones se haya implícita en la manera en que hemos formalizado algunas nociones del metamodelo. Por ejemplo, la noción de *molécula* (que recordemos no es un concepto explícito aportado por Codd) implica la existencia de una forma de establecer qué relaciones forman parte de cada molécula, lo que es parcialmente recogido por la PG-relación propuesta por Codd.

La noción RM/T de *Catálogo*, consistente en un conjunto de relaciones (únicas en cada base de datos RM/T) que indican las ligaduras entre los tipos de relaciones 'de usuario' y sus nombres, los atributos y las relaciones en las que estos se incluyen, etc, comparte de alguna manera la situación de las relaciones grafo. Observemos que en realidad las relaciones grafo y las incluidas en el catálogo desempeñan roles que

pertenecen a un nivel superior de abstracción (es en cierto modo la misma situación que planteábamos con respecto al [E]-dominio en la página 78) por lo que creemos que deberían ser consideradas como tipos de *metarelaciones*, aunque sin duda este planteamiento necesita de un estudio en profundidad.

Sobre las diferencias entre tipos de entidades: aspectos de diseño

Como ya hemos observado, el metamodelo que hemos presentado se centra en los aspectos estructurales, y en particular hemos intentado que se encontrara razonablemente distanciado de los aspectos de diseño de la base de datos. Dicho de otro modo, el metamodelo proporciona los elementos necesarios e imprescindibles para crear una base de datos RM/T (proporciona *qué* contiene una base de datos RM/T, proporciona los productos para modelizar) pero no dice nada del modo utilización de esos elementos para crear tal base de datos (no proporciona *cómo* se construye una base de datos RM/T, no proporciona los procesos para modelizar). Sin embargo, la clasificación en tipos de entidades que propone el modelo (recordemos: núcleos, características, asociativas) está fuertemente ligado al proceso de diseño de una base de datos RM/T y por ello hemos creído conveniente realizar algunas aclaraciones al respecto. En ningún caso nuestra intención es la de proporcionar patrones de diseño mediante el uso de los distintos tipos, sino simplemente tratar de aclarar cuál es la intención de uso con la que aparentemente Codd incluye cada uno de ellos.

Recordemos que en RM/T, y en palabras de Codd, una entidad es característica si «juega un papel subordinado describiendo una entidad de otro tipo» [Cod79, p.411, sec 6]. Date, por su parte, en [Dat92, p. 289, sec 2], especifica que las «características son dependientes de la existencia de la entidad a la que describen», poniendo como ejemplo de característica alguien dependiente de un 'empleado', «como un hijo u otro miembro de la familia». Por otra parte, una entidad es asociativa si «juega un papel superior interrelacionando entidades de otros tipos» según Codd en la misma referencia anterior. Date dice que la función de una entidad asociativa es la de «representar una relación [relationship] varios-a-varios (o varios-a-varios-a-varios, etc.) entre dos o más entidades», poniendo como ejemplo la entidad 'suministro' entendida como asociación de un 'proveedor' y un 'producto'. Si intentamos interpretar estos tipos de entidades en términos de un enfoque E/R, probablemente tanto entidades características como entidades asociativas serían asociaciones para el enfoque E/R. Las preguntas que surgen de manera natural son cómo realizar la distinción

entre entidades características y asociativas y qué beneficios aporta esta distinción.

Posiblemente una de las claves se encuentra en la precisión realizada por Date acerca de la 'dependencia de existencia'[12] en el caso de entidades características. Cuando una entidad dependa para existir de la existencia de otra (lo que dentro de los enfoques E/R es conocido habitualmente como *entidad débil*), será 'natural' modelizarla como entidad característica. Sin embargo este planteamiento necesita de una análisis detallado. Retomando el ejemplo de Date, un 'hijo' sería característica de un 'empleado'. Por tanto estamos estableciendo que la existencia de un 'hijo' es dependiente de la existencia del 'empleado', lo cual es sin duda correcto en el mundo real, en el Universo del discurso. En términos de la base de datos, lo que este tipo de modelización exige es que no se podrá introducir en la base de datos el hijo de un empleado si previamente no se ha introducido el empleado, y lo que es más importante, que si se elimina de la base de datos el empleado, también deben eliminarse sus hijos, *porque lo que se está registrando no es la existencia del hijo –como persona individual, independiente–, sino el hecho de que ese hijo es hijo de un empleado*. Curiosamente los ejemplos que muestra Codd en [Cod79] son menos claros en este sentido. Codd expone que un 'empleo' (puesto de trabajo) es una característica de 'empleado', y que el 'sueldo' es una característica de 'empleo'. En el segundo caso parece claro que la existencia del sueldo depende de la existencia del empleo (si no existe el puesto de trabajo no hay que pagar ningún sueldo por él). Sin embargo en el primer caso parece incorrecto afirmar que la existencia de un puesto de trabajo depende de la existencia del empleado que ocupa ese puesto.

Por otra parte, Date introduce dentro de RM/T un tipo de entidades adicional, las llamadas *entidades designativas* [Dat85, Dat92]. Este tipo de entidades no aparece en la definición de RM/T de Codd y por tanto no pertenece a la clasificación de tipos de entidades de éste. Según Date, estas entidades «tienen un propiedad cuya función es identificar o designar alguna otra entidad relacionada» y «representan una relación [relationship] varios-a-uno entre dos entidades»[Dat92, p. 289,sec 2]. En [Dat85,

[12]El estudio de tipos de relaciones a nivel conceptual para la modelización es un interesante tema de investigación. Podemos encontrar distintas referencias en la literatura que abordan este problema. Por ejemplo [CKSGS94] trata las relaciones de dependencia de existencia en el contexto de las bases de datos federadas o [SD98] considerando la relación de dependencia de existencia como elemento clave dentro del análisis orientado a objetos para la representación de la noción de agregación. La noción de *remisión* en [Dom99] es una perspectiva de estudio de este tipo de relaciones en el ámbito de la Fenomática.

p. 243, sec 6.2], Date excluye a las entidades designativas de la clasificación de entidades, y afirma que «las entidades núcleos, características y asociativas puede ser además designativas». Sin embargo, en [Dat92, p. 289,sec 2] el propio Date dice que «una entidad característica es de hecho un caso especial de entidad designativa; en realidad no es nada más que una entidad designativa que resulta ser dependiente de existencia de la entidad a la que designa». Con este último enfoque podemos revisar los ejemplos anteriores, manteniendo que el 'sueldo' sea característica del 'empleo' y el 'hijo' característica del 'empleado', pero pudiendo situar ahora el 'empleo' respecto del 'empleado' como entidad designativa. Otra ayuda para la interpretación de la diferencia entre entidades designativas y características la proporciona de nuevo Date en [Dat92, p. 299, sec 3], donde se especifica que «las designaciones no causan la introducción de nuevas relaciones. De hecho, una designación se representa simplemente mediante una clave foránea en la relación de la entidad designativa, referenciando la entidad designada». Esta cita sugiere de alguna forma que para que una entidad sea designativa basta añadir un atributo de clave foránea para indicar la designación, mientras que la especificación de una entidad característica sí conlleva la inclusión de una nueva relación. Sin embargo parece que Date tampoco obliga a la inclusión nuevas relaciones para especificar el hecho de ser entidad característica. De hecho ni siquiera obliga a introducir surrogates diferenciados para estas entidades, según se muestra en [Dat92, p. 300, sec 3. Characteristics].

Curiosamente, una de las críticas de Date al modelo E/R es la innecesaria, a su juicio, distinción entre entidades y asociaciones, afirmando que «el mismo objeto puede ser legítimamente considerado por un usuario como entidad y por otro como asociación» [Dat92, p. 304, sec 4], utilizando como ejemplo la noción de 'matrimonio'. Sin embargo, RM/T distingue cuatro tipos (incluyendo las designaciones como un tipo) distintos de entidades, que aun consideradas todas como tales, como entidades, tienen una intención de uso diferente. Es sencillo parafrasear la cita de Date afirmando que en RM/T el mismo objeto puede ser legítimamente considerado por un usuario como entidad núcleo y por otro como entidad asociativa, y por otro como entidad característica, y por otro como entidad designativa.

Para ilustrar estas posibilidades vamos a mostrar un par de ejemplos. Supongamos que nos interesa representar personas que son propietarias de coches. Supongamos también que se imponen ciertas restricciones particulares que hacen que todo coche es propiedad de una y sólo una persona, que una persona puede poseer

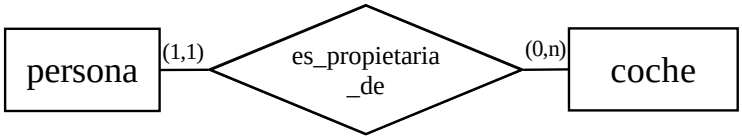


Figura 2.14: Diagrama E/R para el ejemplo de personas propietarias de coches

varios coches, y que no toda persona representada es propietaria de un coche. Esta descripción textual se puede representar de modo bastante estándar en un diagrama E/R como el de la figura 2.14.

Bajo la perspectiva de RM/T, y suponiendo de partida que 'persona' se considera entidad núcleo, existen al menos tres diseños distintos para esta situación mediante el uso de los distintos tipos de entidades:

1. considerar 'coche' como entidad designativa de la entidad 'persona', puesto que cada coche puede designar, identificar a la persona que es su propietario.

2. considerar 'coche' como entidad característica, ya que podemos interpretar que *el hecho de la propiedad del coche* depende de la existencia de la persona que lo compró.

3. considerar 'coche' como entidad núcleo, y considerar de forma adicional una entidad asociativa, supongamos 'propiedad', de tal forma que se representan por separado los coches, las personas, y el hecho de la propiedad de un coche por una persona.

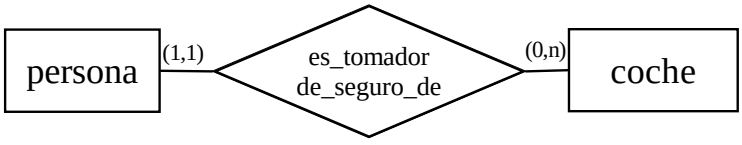


Figura 2.15: Diagrama E/R para el ejemplo de personas tomadores de seguro de coches

Lógicamente cada una de estas soluciones tiene sus ventajas e inconvenientes. La última opción, aparentemente más cercana al diseño E/R de la figura, tiene la

desventaja de la aparición de una entidad "poco natural" (aunque no entraremos a discutir qué significa "natural"). Además, la asociación E/R correspondiente es de tipo uno-a-varios (una persona puede ser propietaria de varios coches) y según las observaciones de Date este tipo de relaciones no deben modelarse con entidades asociativas (que representan relaciones varios-a-varios) sino con entidades designativas (de las cuales recordemos las características son un tipo particular, siempre según Date). Sin embargo la solución no parece tan evidente si pensamos en los posibles atributos descriptores de la asociación E/R. Por ejemplo, el 'impuesto de circulación' no es, desde un punto de vista conceptual, un atributo de la entidad 'coche' (ni tampoco de la entidad 'persona'), sino de la relación de propiedad entre la persona y el coche (ya que sólo se paga este impuesto desde el momento en que la persona compra el coche). Sin embargo, en RM/T, si modelamos 'coche' como entidad designativa o característica, el atributo 'impuesto de matriculación' deberá serlo obligatoriamente de la entidad 'coche'. Observemos que esto es parcialmente debido a que cuando en RM/T se afirma que una cierta entidad es designativa o característica, no hay que olvidar que es designativa o característica *de alguna otra entidad*. Así por ejemplo, al decir que 'coche' es entidad designativa (o característica) en realidad no sólo estamos representando la existencia de un coche, *sino también que ese coche es propiedad de una persona*, de ahí que tenga sentido el atributar la entidad 'coche' con el atributo 'impuesto de circulación'. Esta solución, sin embargo, puede conllevar complicaciones adicionales. Supongamos que tenemos que representar dos asociaciones distintas sobre las mismas entidades, en términos E/R. Por ejemplo supongamos que además de la propiedad del coche tenemos que representar qué personas son los tomadores del seguro de coche, de tal forma que todo coche tiene por tomador del seguro a una y sólo una persona, que una persona puede ser el tomador del seguro de varios coches y puede haber personas que no sean tomador del seguro de ningún coche. Claramente, la representación en un diagrama E/R, que mostramos en la figura 2.15, es prácticamente idéntica a la anterior, con la (importante) salvedad del cambio del vínculo expresado en la asociación entre las entidades 'persona' y 'coche'. Si queremos representar en RM/T de manera simultánea los dos esquemas E/R, y mantenemos la idea de que 'coche' fuera una entidad designativa, nos encontraríamos con una curiosa situación en la que una misma entidad ('coche') es *doblemente designativa*. Esta doble designación es hacia un mismo tipo de entidad (la entidad 'persona'), pero debe mantenerse como doble, puesto en efecto, para un

coche concreto la persona propietaria y la persona tomador del seguro no tienen por qué ser la misma. Esto no parece causar problema al planteamiento a nivel lógico de Date, ya que aparentemente bastaría con añadir dentro de la entidad 'coche' dos claves foráneas distintas hacia la misma entidad 'persona', aunque no parece demasiado coherente con el planteamiento conceptual del modelo RM/T.

Como segundo ejemplo de las distintas posibilidades que aporta RM/T para el diseño de bases de datos, supongamos ahora que lo que se pretende es representar datos de personas junto con sus datos de sus direcciones postales. La gama de tipos de entidades en RM/T nos permite al menos cuatro opciones diferentes de diseño para esta situación. Una primera posibilidad consiste en considerar una entidad núcleo 'persona' que contenga una P-relación donde se almacena la dirección postal para cada persona. Esta solución no es probablemente la más adecuada en caso de que registremos a distintas personas en una misma dirección, ya que obliga a introducir datos repetidos. Más aún, esta solución no permite de ningún modo registrar dos direcciones diferentes (por ejemplo el domicilio particular y la dirección del trabajo) para una misma persona. En estos casos la solución pasa por elevar el status de la noción de 'dirección', convirtiéndola en entidad. Nos encontramos entonces con una segunda opción de diseño, considerando 'persona' como entidad núcleo y 'dirección' como entidad característica de 'persona' (puesto que la dirección describe a –es una característica de– la persona). Esta solución de nuevo no es óptima. En primer lugar es "poco natural" considerar la 'dirección' como dependiente de la existencia de la 'persona', y en segundo lugar, y quizá más importante, con este diseño no se podrían registrar dos personas en la misma dirección, puesto que la C-relación correspondiente dentro de la molécula característica tendría como clave principal el E-atributo correspondiente a la dirección, y en este, obviamente, no pueden aparecer repetidos. Esto nos lleva a la tercera opción, que pasa por considerar a 'persona' y 'dirección' como entidades núcleo, pero de tal forma que además 'persona' es entidad designativa hacia 'dirección'. De esta manera cada persona señala, designa su dirección. Esta solución facilita la representación de la misma dirección para dos personas distintas, e incluso permite representar direcciones sin ninguna persona asignada. Sin embargo esta solución todavía no recoge la posibilidad de registrar dos direcciones diferentes para una misma persona. Si este es un requisito obligado, entonces es necesario representar 'persona' y 'dirección' como entidades núcleo independientes, e introducir una nueva entidad, de tipo asociativo, que relacione las personas y sus direcciones.

Analizando a posteriori los cuatro casos presentados, podemos ver una clara analogía con los cuatro casos habituales de cardinalidad que se distinguen en las asociaciones dentro de los enfoques E/R, concretamente los tipos uno–a–uno, uno–a–varios, varios–a–uno y varios–a–varios. Esto sugiere que una posible utilidad del modelo RM/T sería la de paso intermedio para la traducción de esquemas E/R en modelos relacionales. La ventaja de los modelos E/R es su fácil comprensión por el usuario, aunque por el contrario tienen la desventaja de ser modelos poco rigurosos. Por su parte, el modelo relacional se fundamenta en una sólida base matemática, rigurosa por tanto, pero es más arduo de aplicar para el diseño. La utilización de esquemas E/R para diseño y su traducción posterior al modelo relacional tiene como mayor inconveniente la pérdida de significado en el proceso de traducción. El estudio del modelo RM/T como modelo intermedio entre modelos E/R y el modelo relacional se presenta como una línea de trabajo abierta.

Subtipos y agregaciones en RM/T

El último aspecto relevante del modelo RM/T que no hemos abordado en el metamodelo es el relativo a la noción de subtipo. Codd introduce esta idea dentro de RM/T simplemente afirmando que «un tipo de entidad e_1 se dice que es un subtipo de un tipo de entidad e_2 si todas las entidades de tipo e_1 son necesariamente entidades del tipo e_2». En los años en que se ideó el modelo RM/T las ideas relativas a las nociones de ISA, subclase, subtipo, etc, se encontraban en pleno proceso de gestación, proceso que sin duda no ha terminado aún. Existe abundante literatura que trata el tema (véanse a modo de ejemplo [SS77, Bra83, DB90, TCGB90, LP91, ACS98, AKHS00, AK00, PW00], sin que en ningún caso ésta sea una lista exhaustiva) aunque probablemente estamos lejos de alcanzar un punto de vista que sea aceptado de forma general. En el caso particular de RM/T el enfoque de Codd trata la idea de subtipo de forma exclusivamente conjuntista, a un nivel extensional. Aunque este planteamiento es aparentemente simple, dentro de RM/T tiene implicaciones importantes. En particular, la caracterización directa de que e_1 sea subtipo de e_2 supone que todos los surrogates que aparecen en la E-relación de la molécula que representa a e_1 deben aparecer también en la E-relación de la molécula que representa a e_2. Observemos que estamos obligados a referir a los surrogates y no a los valores de E-atributo, puesto que debido a las diferentes restricciones impuestas en el metamodelo, los nombres de E-atributo de

dos E-relaciones cualesquiera son distintos[13] , lo que impide que un mismo valor de E-atributo aparezca en dos E-relaciones diferentes. Esta idea también es expuesta por Codd, si bien en forma de regla, la *regla de integridad de subtipo*: «siempre que un surrogate (sea s) pertenezca a la E-relación de una entidad de tipo e, s debe pertenecer también a la E-relación para cada tipo entidad del cual e es subtipo». Aunque hemos creído más conveniente no incluir el concepto explícitamente en el metamodelo, debido a la complejidad subyacente al mismo, hubiéramos podido introducir la noción de subtipo (y su regla asociada) en nuestra formalización mediante el concepto:

submolécula_de_molécula_entidad ∗molécula_entidad tal que cada surrogate de valor_de_E-atributo_de_E-relación de la molécula es además surrogate de valor_de_E-atributo_de_E-relación otra molécula_entidad∗	int.

Observemos que en el metamodelo tal y como está planteado en las secciones anteriores, no puede existir un surrogate que aparezca simultáneamente en dos E-relaciones distintas. La introducción de la noción de submolécula cambiaría esta situación, pero además podría ocurrir que dos E-relaciones contuvieran surrogates comunes sin tener que pertenecer a dos moléculas que sean la una submolécula de la otra, simplemente porque ambas moléculas tengan una submolécula común. Esto es admitido explícitamente por Codd: «un tipo de entidad puede ser generalizado por inclusión en dos o más tipos de entidad».

Otra implicación particular de este planteamiento es la llamada por Codd *regla de la propiedad de herencia*, según la cual «dado cualquier subtipo e todas las propiedades de sus tipos padre son aplicables a e». Puesto que como acabamos de ver una molécula puede ser submolécula de dos o más moléculas, un tipo de entidad puede heredar propiedades de más de un subtipo: esta situación, que ha sido tambíen estudiada en la literatura y que tendría que ser analizada con detalle en el caso de RM/T, es conocida habitualmente como *herencia múltiple*.

La última implicación básica relativa a la noción de subtipo es que Codd res-

[13]Date presenta en [Dat85, p.267, sec 6.10] una interesante idea relacionada con la nominación de E-atributos´, la noción de *alias*. La idea es que si un tipo de entidad es subtipo de otra (supongamos 'director' subtipo de 'empleado') es posible referir al E-atributo de la E-relación del subtipo también vía el E-atributo del supertipo (por ejemplo el nombre 'director.empleadoç' sería alias del nombre 'directorç')

tringe las posibilidades de subtipos puesto que «un subtipo de un tipo de entidad característica es también característica; un subtipo de un tipo de entidad núcleo es también núcleo; un subtipo de un tipo de entidad asociativa es también asociativa» [Cod79, p. 411, sec 6] (Date reafirma esta idea en [Dat92, p. 294, sec 2. Subtypes and Supertypes]). Las implicaciones de estas restricciones, que tendrían que ser enunciadas explícitamente si quisiéramos incluirlas en este metamodelo son varias, y distan mucho de ser triviales. Por ejemplo, si un tipo de entidad característica es subtipo de otra, el tipo de entidad caracterizada aparentemente debería ser el mismo, pero esto no es impuesto explícitamente por el modelo. En términos utilizados en el metamodelo, si una molécula característica es submolécula de otra el E-atributo no-principal en las dos C-relaciones correspondientes debería ser el mismo. Se puede realizar una observación similar para las entidades asociativas. A la vista de estas puntualizaciones, parece claro que la idea de subtipo dentro de RM/T así como sus relaciones con la noción de molécula que hemos introducido necesitan un análisis pormenorizado.

A pesar de las dificultades inherentes a la noción de subtipo, RM/T utiliza esta noción no sólo como 'primitiva' sino como medio para la definición de estructuras jerárquicas más complejas para los tipos de entidad. En particular, Codd introduce las nociones de generalización incondicional y generalización alternativa.

La idea de *generalización incondicional* es la de considerar todos los subtipos $x_i, (i = 1 \ldots n)$ de un tipo de entidad y dado (junto con los subtipos de sus subtipos –si los hubiera–, etc.) de forma global, como un todo. Desde un punto de vista estrictamente conjuntista, que como hemos observado es la idea elemental de subtipo en RM/T, la generalización incondicional está tratando el caso en que $\bigcup_{i=1}^{n} x_i \subseteq y$. Aunque no tienen relevancia desde un punto de vista conceptual sí que es importante, desde el punto de vista extensional, distinguir algunos casos dentro de la generalización incondicional. En primer lugar tenemos el caso en que $x_i \bigcap x_j = \emptyset, \forall i, j$, es decir, la generalización se obtiene a partir de una unión disjunta, en la que todas las entidades de los subtipos x_i de y son distintas. Por ejemplo podríamos tener los tipos de entidad 'perro', 'gato' y 'pájaro' generalizados al tipo de entidad 'mascota'. En este caso no se obliga a que toda entidad 'mascota' sea entidad de uno de los subtipos. Si se impone esta restricción, habitualmente se dice que los x_i constituyen una *partición* de y. En segundo lugar tenemos el caso en que $x_i \bigcap x_j \neq \emptyset,$ para algún i, j, es decir, la generalización se obtiene de una unión no disjunta, y por tanto puede haber

entidades comunes en dos subtipos de y. Por ejemplo podríamos tener los tipos de entidad 'padre' y 'hermano' generalizadas al tipo de entidad 'pariente'. De la misma forma que en el caso anterior, no se obliga a que toda entidad 'pariente' sea entidad de uno de los subtipos. En el caso de que se imponga esta restricción, decimos que los x_i constituyen un *cubrimiento* de y, por ejemplo en el caso de los tipos de entidad 'miembro_pdi', 'miembro_pas' y 'alumno' generalizados a 'miembro_universidad'.

Por otro lado, la idea de *generalización alternativa* es la de que «un tipo entidad puede ser generalizado dentro de dos o más tipos alternativos». Por ejemplo las entidades del tipo de entidad 'cliente' podrían ser entidades o bien del tipo 'empresa', o bien del tipo 'institución' o bien del tipo 'individual' (en función de si el cliente es una empresa, una institución o un individuo). Observemos que este es un tipo de generalización distinto del anterior, ya que en este caso ocurre que lo generalizado no es subtipo de la generalización. En el ejemplo, 'cliente' no es es subtipo de 'empresa' (ni de 'institución' ni de 'individuo), ya que no todo cliente es empresa. Además, la utilidad principal de este nuevo tipo de generalización ocurre cuando tampoco se verifica la noción de subtipo en sentido inverso. En el ejemplo, debe entenderse que no toda empresa es cliente (y análogamente con institución e individuo), ya que si así fuera, tendríamos el tipo 'empresa' como subtipo de 'cliente', y podríamos considerar el tipo 'cliente' como generalización incondicional de sus subtipos. Observemos que de hecho los dos tipos de generalización (incondicional y alternativa) son compatibles para un mismo contexto, para un mismo conjunto de tipos de entidad. En el ejemplo anterior, los tipos de entidad 'empresa', 'institución' e 'individuo' pueden generalizarse incondicionalmente por ejemplo a un tipo de entidad 'unidad_legal', lo que implica que por ejemplo toda empresa es una unidad legal (mientras que en la generalización alternativa, insistimos, no todo cliente es empresa, sino que es o empresa, o institución, o individuo). Desde un punto de vista conjuntista, lo que estamos considerando en este caso es que $y \subset \bigcup_{i=1}^{n} x_i$[14]. Observemos el detalle de que el contenido como subconjunto de y en $\bigcup_{i=1}^{n} x_i$ ha de ser un contenido estricto, puesto que si pudiera darse el caso de que $y = \bigcup_{i=1}^{n} x_i$ tendríamos automáticamente que $x_i \subset y, \forall i$ y por tanto todos los x_i serían subtipo de y, lo que nos lleva al caso de generalización incondicional. De nuevo nos

[14]En un nivel conceptual la unión de entidades no tiene por qué obligatoriamente ser considerada entidad. Por ello no podemos hablar con propiedad de que y sea un subtipo de $\bigcup_{i=1}^{n} x_i$, y sólo podemos tratarlo a nivel conjuntista.

encontramos con dos posibilidades distintas para la generalización alternativa. En un primer caso, puede suceder que $(x_i \bigcap x_j) \bigcap y = \emptyset, \forall i, j$ (un sub-caso particular en que se da esta situación es cuando $x_i \bigcap x_j = \emptyset, \forall i, j$). Esto significa que toda entidad de tipo y es de uno y sólo uno de los tipos x_i: esta situación se identifica habitualmente diciendo que la inclusión (de y en $\bigcup_{i=1}^{n} x_i$) es disjunta. En el ejemplo anterior relativo a clientes la inclusión es disjunta (ya que en este ejemplo particular ocurre en concreto que 'empresa' $\bigcap$ 'institución' $= \emptyset$, 'empresa' $\bigcap$ 'individuo' $= \emptyset$ e 'institución' $\bigcap$ 'individuo' $= \emptyset$). En el segundo caso, sucede que $(x_i \bigcap x_j) \bigcap y \neq \emptyset,$ para algún i, j (con lo que en consecuencia, $x_i \bigcap x_j \neq \emptyset,$ para algún i, j) con lo que nos encontramos con una inclusión no disjunta. Por ejemplo, podríamos considerar la generalización alternativa del tipo de entidad 'animal_carnívoro' a los tipos de entidad 'animal_terrestre', 'animal_acuático' o 'animal_volador'.

Como último apunte acerca de las ideas relativas a las generalizaciones incondicional y alternativa, observemos que éstas, por sí mismas, no cambian la estructura intensional del modelo, aunque pueden tener importancia en otros sentidos. Por ejemplo, durante el diseño de la base de datos habrá que decidir la aparición o no de estas relaciones, o, desde un punto de vista operacional, el tipo concreto de generalización que se considere puede influir por ejemplo en el modo en que realiza la gestión de altas–bajas–modificaciones en la base de datos RM/T.

El último tipo de abstracción considerado por Codd en el modelo RM/T es la noción de *agregación por cubrimiento* (*cover aggregation*). Esta noción se corresponde de manera más o menos fiel con la noción de *agregación*, también conocida como relación *ser–parte–de*, noción que es incluida actualmente en distintos lenguajes de modelización. Sobre esta noción existen distintos estudios en la literatura, algunos de ellos muy recientes, como por ejemplo [BHS01, OHSB01]. En RM/T la relación todo–parte se denomina agregación por cubrimiento porque Codd ha de distinguirla de lo que él llama *agregación cartesiana*. Para Codd tanto el hecho de agrupar un conjunto de propiedades en una sola estructura (las moléculas de nuestro metamodelo) como el hecho de considerar conjuntamente los asociados de una molécula asociativa son casos de agregación, denominada genéricamente cartesiana. La agregación por cubrimiento aparece porque, por ejemplo, «una flota de barcos es ciertamente un agregación de algún tipo. Sin embargo no es ni una abstracción por agregación cartesiana ni una abstracción por generalización (ya que los barcos no son ni instancias ni subtipos de flotas)» [Cod79, p. 422, sec 12]. Podemos consi-

derar otro ejemplo de agregación por cubrimiento en la relación entre profesores y áreas de conocimiento: entre el tipo de entidad 'profesor' y 'área_de_conocimiento' no existe relación de subtipo, y sin embargo las entidades en 'área_de_conocimiento' pueden entenderse como una agregación de entidades 'profesor'. Observemos que otras soluciones de diseño alternativas no capturarían fielmente el significado pretendido. Por ejemplo, podríamos definir el tipo de entidad 'profesor' como entidad designativa hacia 'área_de_conocimiento', pero lo que estaríamos representando de manera directa es la relación entre *un* profesor y *un* área de conocimiento, y no el hecho de el área de conocimiento se obtiene como agregación de profesores. El ejemplo de flotas y barcos usado por Codd sirve para dar nuevas razones que defienden la existencia de la agregación cover. En primer lugar, podría interesarnos registrar barcos independientes, no pertenecientes a ninguna flota, y en segundo lugar se pueden considerar flotas no–homogéneas, es decir, considerar una flota como un conjunto de barcos más un conjunto de aviones más un conjunto de camiones (imagínese la situación de una empresa de transporte donde se registra el conjunto de medios necesarios para transportar una mercancía de un punto a otro: para un caso concreto podrían ser necesarios dos camiones, un avión y un barco, por ejemplo). Cualquiera de estas posibilidades, si bien no impide estrictamente otros planteamientos de diseño (como la consideración de entidades designativas) sí que dificulta su interpretación correcta. A pesar de que Codd expone la agregación por cubrimiento como un elemento más del modelo RM/T, entendemos que es un tipo de relación que necesita de un estudio en profundidad, lo que es reforzado por el hecho de que Date no menciona la agregación por cubrimiento en ninguna de sus dos revisiones de RM/T [Dat85, Dat92].

Capítulo 3

Metamodelización de formalismos de comportamiento

En el capítulo anterior se han presentado las ideas fundamentales de la técnica de metamodelización NÓESIS. Para ilustrar dichas ideas se ha mostrado un metamodelo del modelo RM/T de bases de datos. Puesto que en efecto RM/T es un modelo para el desarrollo de bases de datos, los conceptos que se manejan son fundamentalmente estructurales, es decir, son conceptos que permiten la modelización de estructuras de datos. En muchas ocasiones se refiere a este tipo de conceptos también con el calificativo de *estáticos*, en contraposición a conceptos *dinámicos* que son usados por lenguajes y/o formalismos cuyo objetivo fundamental es la representación de aspectos de comportamiento de un determinado sistema. Al igual que hemos indicado en el Capítulo 1 de este trabajo, de ahora en adelante, y abusando en cierto modo del lenguaje natural, utilizaremos el término *formalismo de comportamiento* para referirnos a formalismos que han sido creados para modelar aspectos dinámicos. Por ejemplo, el formalismo Statecharts [Har87] es un formalismo de comportamiento que fue creado como una ampliación de los clásicos diagramas de estado-transición, y cuyo principal objetivo es facilitar la especificación de sistemas reactivos. Actualmente es un formalismo cuyo uso se ha extendido enormemente, pues ha sido adoptado por diversas metodologías orientadas a objeto para la especificación de comportamiento.

Uno de los objetivos del presente trabajo es el de desarrollar un metamodelo del formalismo Statecharts de tal forma que el metamodelo capture de manera fiel la esencia del formalismo, en particular representando explícitamente aquellos elementos que caracterizan al formalismo como lenguaje para la representación de aspectos

de comportamiento. Un análisis detallado de estas características nos han conducido a la necesidad de establecer un marco conceptual adecuado que nos permitiera tratar el problema desde una perspectiva de metamodelización. En particular, este análisis nos ha llevado al desarrollo de una arquitectura que proporciona un marco conceptual general para la representación de comportamiento, siendo de hecho aplicable no sólo a Statecharts sino a otros formalismos de comportamiento, e incluso como marco para la interpretación de las nociones esenciales de representación de comportamiento en otros niveles de abstracción. Este capítulo está dedicado a presentar de forma detallada esta arquitectura, a la que hemos denominado *arquitectura* NÓESIS, ilustrándola a través de distintos ejemplos e indicando cuál es su interpretación en función del nivel de abstracción bajo el que se considere.

3.1. Formalismos de comportamiento: estática y dinámica

La especificación y/o modelización de los aspectos relativos al comportamiento de los sistemas es una de las tareas fundamentales que es necesario llevar a cabo durante el desarrollo de un sistema software. Para realizar esta tarea, existen diferentes técnicas de modelización que permiten la representación de las características dinámicas de sistemas. Algunas de estas técnicas han de ser denominadas con mayor precisión *formalismos*, ya que están fundadas en un enfoque formal, habitualmente matemático. Ejemplos claros de este tipo de técnicas son el ya citado formalismo Statecharts y el formalismo *Redes de Petri* [Pet81], éste último utilizado con especial énfasis en el área de sistemas de tiempo real. Otras técnicas han sido desarrolladas de modo algo más informal, como por ejemplo los Diagramas de Flujo de Datos, los Diagramas de Secuencias o los Diagramas de Interacción. Todas y cada una de estas técnicas están orientadas hacia un ámbito de modelización específico. Sin embargo, hay una característica esencial que prácticamente todas estas técnicas comparten, y es que con ellas se pueden elaborar modelos a los que se les puede dar una interpretación dinámica. Más en concreto, la gran mayoría de los formalismos de comportamiento permiten crear modelos mediante una representación diagramática, y por tanto disponen de una serie de símbolos que podríamos llamar *estructurales*, ya que proporcionan la estructura del modelo. Pero además, y puesto que estos formalis-

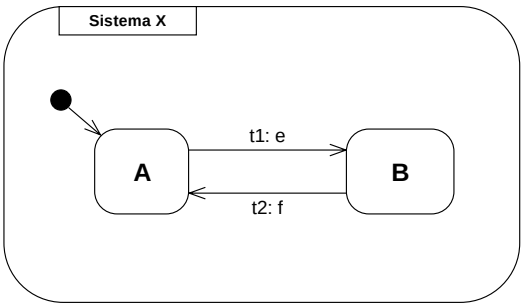


Figura 3.1: Statechart para un Sistema X: Vista Estática

mos están especialmente diseñados para la representación de comportamiento, esos diagramas estructurales deben interpretarse desde un punto de vista dinámico, en el sentido de la posibilidad de variación del significado o de la valoración de algunos elementos de tales diagramas. Esta distinción, fundamental para la metamodelización de este tipo de formalismos, nos ha llevado a distinguir dos aspectos diferenciados a la hora de tratar tales técnicas. Hablaremos de *Vista Estática* de un formalismo de comportamiento cuando nos refiramos a las nociones de tipo estructural que el formalismo proporcione para modelar, y de *Vista Dinámica* cuando nos refiramos a la interpretación dinámica que tenemos que dar a los modelos, según indique el propio formalismo. Es necesario aclarar que en este análisis se están manejando en realidad dos tipos de comportamiento: por una parte el comportamiento que se representa a través de los modelos creados con el formalismo, y por otra parte el comportamiento de los modelos creados con el formalismo, entendido éste como las variaciones a las que los distintos elementos del modelo están sometidos con el objetivo de que representen comportamiento del sistema modelizado. Lo que un metamodelo debe capturar es el segundo tipo de comportamiento, puesto que éste es el que se encuentra en el mismo nivel de abstracción que el propio formalismo. Estas ideas, que dirigirán de alguna manera el resto del trabajo, se entremezclan en algunos casos con las nociones de sintaxis y semántica de un formalismo.

Para aclarar el concepto clave de distinción entre Vista Estática y Vista Dinámica, vamos a estudiar un pequeño ejemplo utilizando Statecharts. Supongamos un cierto Sistema X que puede encontrarse solamente en dos situaciones, A y B, y que inicialmente debe estar en la situación A. Si en dicha situación se produce una cierta acción e, el sistema se moverá a la situación B, y si entonces se produce una acción f el sistema volverá a la situación A. Esta descripción textual se encuentra al ni-

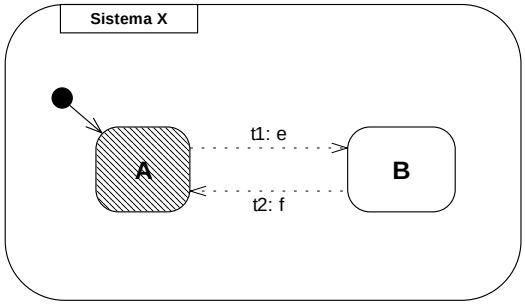


Figura 3.2: Statechart para el Sistema X con estado A activo

vel de aplicación, es decir, al nivel en el que se está representando el sistema, pues constituye de hecho un *modelo textual* del mismo.

La Figura 3.1 es un statechart (por tanto un modelo visual, gráfico) del mismo Sistema X, y representa de manera conjunta *todas las eventuales situaciones* en las que el sistema se puede encontrar. Para crear el modelo se han utilizado diversos conceptos de Statecharts, tales como *estado* (A y B), *transición* (t1 y t2), *evento* (e y f) o *conector por defecto*. En efecto este modelo captura todas las eventuales situaciones, puesto que en él se encuentra representado el estado B, en el que *potencialmente podría no residir nunca el sistema* (simplemente por no haberse producido el evento e). Debido a ello, lo que un diagrama statechart captura es la *estructura estática del comportamiento* del sistema, y por tanto el conjunto de conceptos Statecharts que son necesarios para crear un diagrama statechart forman parte de la *Vista Estática* del formalismo. Esta vista contrasta con los conceptos Statecharts que deben manejarse cuando se describe el *comportamiento real del propio statechart*.

En una hipotética ejecución del statechart que modela el Sistema X, la existencia del conector por defecto haría que el estado inicialmente activo fuera el estado A. Sin embargo, el diagrama statechart 3.1 no puede reflejar de manera explícita esta situación concreta (excepto por la interpretación que el lector, conocedor de los símbolos gráficos que usa el formalismo, hace del diagrama). Una posible representación de la situación inicial aparece en la figura 3.2. Para la elaboración de este diagrama hemos ideado una notación gráfica especial para representar por ejemplo un *estado activo* (a través de una caja sombreada) o una *transición no disparable* (a través de una línea punteada), ya que el formalismo Statecharts no proporciona, al menos de manera "oficial", ningún símbolo para representar este tipo de conceptos[1]. La razón

[1]A este respecto es necesario observar que en algunas de las referencias relacionadas con la

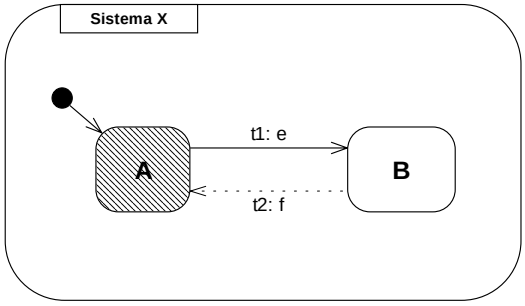


Figura 3.3: Statechart para el Sistema X con transición t1 disparable

fundamental para la no existencia de estos símbolos es que en efecto este tipo de conceptos (*estado activo*, *transición no disparable*) no pertenecen a la estructura de los diagramas statechart, sino al comportamiento de dichos diagramas, es decir, a la *Vista Dinámica* del formalismo Statecharts.

Continuando con el análisis del comportamiento del Sistema X, si estando en la situación que describe la figura 3.2 se produce el evento e, el statechart necesariamente ha de cambiar. La nueva situación se refleja en la figura 3.3, donde se ha representado el hecho de que la transición t1 pasa a ser disparable a través de una línea continua. De nuevo, en esta descripción textual aparecen conceptos que pertenecen a la Vista Dinámica del formalismo, como *ocurrencia de evento* o *transición disparable*. La situación del statechart en este momento es una situación "virtual", en el sentido de que el significado de la noción *transición disparable* hace que el sistema no esté en tal situación de manera estable, sino que automáticamente la transición se dispara (pasando a ser de nuevo no disparable) y el estado activo pasa a ser B, como se muestra en la figura 3.4. De manera análoga se podría describir ahora el comportamiento del statechart con respecto a la transición t2.

La primera idea fundamental que el ejemplo aclara es que cuando se crea un modelo de un sistema utilizando Statecharts se está haciendo referencia a dos comportamientos: por una parte el comportamiento del sistema que esta modelando y por otra el comportamiento del statechart que modela el sistema. Cuando se describe el comportamiento del sistema (es decir, cuando se crea un modelo/diagrama statechart) se utilizan conceptos como *estado* o *transición*, mientras que cuando se describe el comportamiento del diagrama statechart se utilizan conceptos muy re-

definición de Statecharts (véase, por ejemplo, [HN96]) el símbolo 'caja sombreada' sí que se utiliza (sin definición explícita) para representar la noción de estado activo.

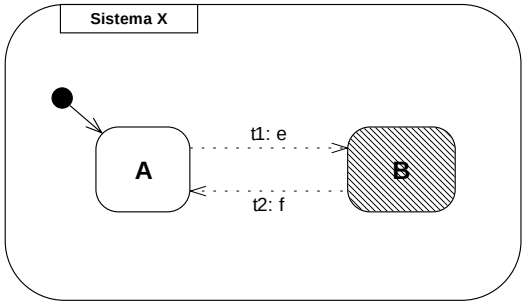


Figura 3.4: Statechart para el Sistema X con estado B activo

lacionados, pero diferentes, tales como *estado activo* o *transición disparable*. Esta diferencia, que hemos podido detectar a través de un ejemplo muy sencillo, necesita de un análisis mucho más detallado cuando se considera el conjunto de características de las que dispone Statecharts, tales como *estados AND* y *estados OR*, o los distintos tipos de *conectores* (*AND*, *OR*, *History*, etc). Puesto que uno de los principales objetivos de este trabajo es la elaboración de un metamodelo del formalismo, debemos recoger en él toda la potencia expresiva de Statecharts.

Es posible extraer del ejemplo del Sistema X una segunda conclusión referida al comportamiento de un statechart. Esta conclusión tiene en realidad dos partes. Por una parte, el comportamiento de un statechart comienza a partir de una situación inicial. La sencillez del ejemplo que estamos manejando hace que en este caso concreto la situación inicial se limite a que el estado A debe ser activo, y este hecho viene representado por la existencia en el diagrama del conector por defecto. Sin embargo, en un statechart real (y por tanto mucho más complejo en general) pueden existir multitud de características para las que sea necesario fijar su valor inicial, tales como valores de variables, o condiciones y eventos primitivos. Para este tipo de características el formalismo no proporciona *a priori* forma gráfica de representar no ya sus valores iniciales, sino tan siquiera la *necesidad de proporcionar* tales valores. Sin embargo, parece claro que la obtención de los valores iniciales es un requisito indispensable para la comprensión del comportamiento del statechart. Imaginemos una herramienta CASE que realice tareas de simulación de comportamiento de sistemas mediante la animación de diagramas statechart. En el momento del inicio de la simulación, la herramienta CASE podrá probablemente deducir una parte de los valores iniciales del propio diagrama, pero la herramienta deberá estar programada para preguntar al usuario por aquellos valores iniciales no deducibles del diagra-

ma. Entendemos que si en la especificación del lenguaje Statecharts (dicho de otro modo, en el metamodelo del formalismo) se hacen explícitas aquellas nociones que son susceptibles de necesitar la asignación de valores iniciales (con independencia de quien proporcione tales valores: operador humano, datos obtenidos de un sensor, datos derivados de otros integrados en el modelo...) el formalismo se hace más comprensible, facilitando, por ejemplo, el diseño de herramientas CASE. Si pensamos en términos de los diagramas con los que estamos modelando el Sistema X, estamos haciendo patente que el formalismo debe proporcionar un método para pasar de la figura 3.1, es decir, de un diagrama que pertenece a la vista estática, a la figura 3.2, el cual es un diagrama que pertenece a la Vista Dinámica.

Por otra parte, una observación similar, mas si cabe de una importancia más crucial, puede hacerse para la relación entre los diagramas 3.2, 3.3 y 3.4 del ejemplo. Cada uno de los diagramas 3.2 y 3.4 representa, utilizando la terminología que proporciona Statecharts, un *status*, es decir, una instantánea detallada de la situación del sistema. El cambio de un status al siguiente (que en el ejemplo concreto se limita a un cambio de estado) se corresponde con el concepto Statecharts de *paso*. Esta es la noción fundamental para describir el comportamiento de un statechart: un paso es la obtención del siguiente status a partir del anterior y (eventualmente) de información adicional. Como ocurría con la determinación de la situación inicial (es decir, del status inicial) en el ejemplo la descripción del paso es extremadamente sencilla; sin embargo, como veremos más adelante, la descripción de qué es un paso constituye la mayor dificultad para la especificación detallada del metamodelo NÓESIS de Statecharts. Tanto es así que será prácticamente indispensable la introducción de una cierta noción de *paso intermedio* que facilite tal descripción. Esta idea ya aparece, si bien de forma somera, en el ejemplo del Sistema X. Como ya se ha dicho, la situación que representa el diagrama 3.3 es una situación "no estable". Lo que sucede en realidad es que este diagrama no representa un status, puesto que precisamente esta noción es la que se corresponde con la de situación "estable". Sin embargo, para la plena comprensión del procedimiento de cambio de paso se hace imprescindible considerar diagramas tales como el de la figura 3.3, puesto que es en él en el que aparece explícitamente la noción de *transición disparable*: la determinación, en sentido amplio, de qué transiciones son disparables en un statechart constituye quizá la tarea más importante, y con seguridad la más compleja, de todas las relacionadas con la descripción y comprensión del comportamiento de los

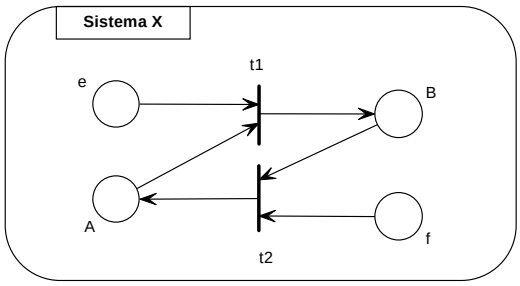


Figura 3.5: Red de Petri para el Sistema X: Vista Estática

statecharts.

Para concluir esta sección, vamos a mostrar cómo las ideas expuestas no son exclusivas de Statecharts, a través de un modelo de nuestro ejemplo creado utilizando otro formalismo de comportamiento. En concreto, vamos a considerar Redes de Petri, que es otra técnica ampliamente utilizada para el desarrollo de modelos dinámicos, especialmente en el área de sistemas de tiempo real. En la sección 3.2 trataremos con algo más de detalle este formalismo, limitándonos ahora a las nociones mínimas necesarias para la comprensión del ejemplo. En la figura 3.5 aparece representado el Sistema X de nuestro ejemplo a través de una red de Petri.

En una red de Petri, los *lugares* representan determinadas condiciones, que en el caso de que se cumplan, dan lugar a la ejecución de determinados eventos, modelados a través de *transiciones*. En el ejemplo del Sistema X modelado a través de una red de Petri, las dos situaciones A y B en las que puede estar el sistema, así como las acciones e y f que se pueden producir, están modeladas cada una de ellas a través de un lugar. El cambio de una situación a otra está representado por transiciones.

Obviamente no estamos afirmando aquí que esta sea la mejor red de Petri posible para modelar el Sistema X, ni mucho menos discutiendo si la red de Petri de la figura 3.5 es equivalente (o no) al statechart de la figura 3.1. De hecho, como ya se ha comentado en los Preliminares, la comparación entre distintos formalismos es una de las utilidades de la metamodelización, y por tanto estaríamos abordando un problema de una magnitud muy diferente. En este momento simplemente vamos a utilizar el ejemplo para mostrar cómo las nociones de Vista Estática y Vista Dinámica de un formalismo de comportamiento, que acabamos de identificar para Statecharts, son también son fácilmente observables en Redes de Petri.

En una red de Petri se distingue el *grafo* que constituye la red del *marcado* de

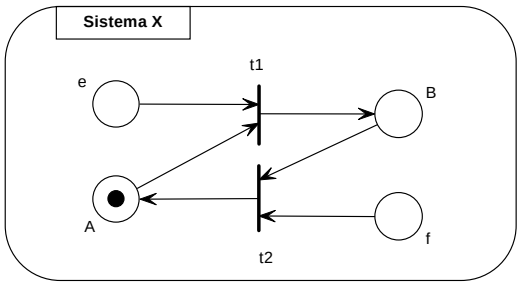


Figura 3.6: Red de Petri para el Sistema X con lugar A marcado

dicha red. La figura 3.5 es un grafo no marcado: de forma análoga a lo observado para Statecharts, esta figura representa todas las eventuales situaciones en que puede residir el sistema, y para ello utiliza las nociones de *lugar*, *transición* y *arco*. Todas estas nociones pertenecen por tanto a la Vista Estática del formalismo Redes de Petri. Una situación concreta del sistema se modelaría a través de un marcado concreto de la red. Supongamos, por ejemplo, el marcado de la figura 3.6. Su interpretación sería que la condición "estar en la situación A" se cumple, y por tanto se podría establecer un paralelismo entre este diagrama y la figura 3.2. Así la nociones de *marca de un lugar* y de *marcado de una red* pertenecen a la Vista Dinámica del formalismo Redes de Petri.

En este ejemplo, de nuevo podemos observar la necesidad de fijar la situación inicial (el *marcado inicial* para la red de Petri), o, en términos de los diagramas, la necesidad de pasar de un grafo no marcado (figura 3.5) a un grafo marcado (figura 3.6). Además, podemos observar como, también en Redes de Petri, el mecanismo de disparo de transiciones constituye una de las tareas básicas para la descripción de los modelos. En efecto, si consideramos el marcado de la figura 3.7, vemos reproducida la situación de la figura 3.3, puesto que tal situación es una situación "virtual": el mecanismo de disparo de una transición en Redes de Petri obliga a que en la situación de la figura 3.7, automáticamente se dispare la transición `t1`, se eliminen las marcas de los lugares `A` y `e` y se coloque una marca en el lugar `B`. Por tanto, y de forma análoga a como sucedía en Statecharts, nos encontramos ante la necesidad de representar en un hipotético metamodelo de Redes de Petri, el paso de una red de Petri marcada a otra, concepto fundamental que pertenece a la Vista Dinámica de este formalismo.

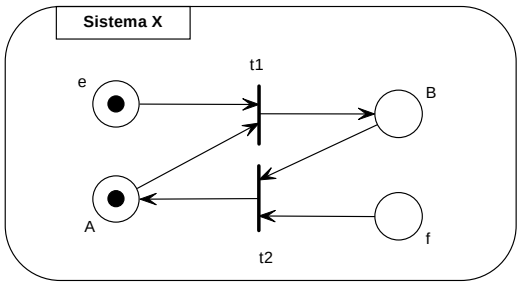


Figura 3.7: Red de Petri para el Sistema X con lugares A y e marcados

3.2. La arquitectura Nóesis para la representación del comportamiento

Como acabamos de ver en la sección anterior, un análisis detallado de algunos formalismos de comportamiento nos han llevado a la conclusión de la existencia de dos vistas diferentes dentro de cada formalismo: la Vista Estática, referida a las nociones estructurales que proporcione para modelar, y la Vista Dinámica, que abarca los conceptos de comportamiento de los modelos creados. Cada una de estas vistas, por lo tanto, hacen referencia a aspectos que pueden observarse para cualquier formalismo de comportamiento. Parece entonces razonable considerar estas ideas desde un mayor nivel de abstracción, de manera que se disponga de un marco genérico para la representación de las características de comportamiento con independencia de cuál sea el formalismo concreto que se esté considerando.

En esa línea hemos ideado la *arquitectura* NÓESIS para la representación de comportamiento. Esta arquitectura puede ser utilizada como guía general para el análisis de las características de comportamiento de un determinado sistema o de un determinado formalismo. Como veremos de inmediato, una de las principales ventajas de la arquitectura es que puede utilizarse con independencia de cuál sea el nivel de abstracción en el que el analista (o meta–analista en su caso) se encuentre situado. En una primera aproximación, la arquitectura NÓESIS se compone de dos capas o niveles y de dos transformaciones que relacionan dichos niveles: en la figura 3.8 aparece una posible representación gráfica de la arquitectura.

El primer nivel, denominado **Nivel de Independencia de Status**, representa todas aquellas nociones que pueden identificarse sin necesidad de hacer referencia un status concreto. Es necesario aclarar que hemos decidido utilizar la palabra 'status'

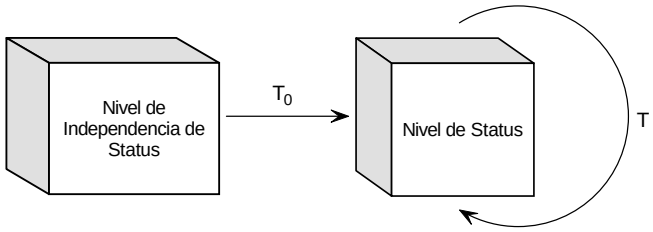


Figura 3.8: Representación gráfica de la Arquitectura NÓESIS

en lugar de la palabra 'estado' para evitar posibles ambigüedades con respecto al uso de estas palabras para el caso particular del formalismo Statecharts. En este formalismo, como ya se ha visto, el término 'estado' se utiliza para designar uno de sus conceptos fundamentales, y si bien es cierto que, como no podía ser de otra manera, está íntimamente ligado a la idea de "cómo se encuentra el sistema", es el término 'status' el que se refiere de manera precisa a la situación global en que se encuentra el sistema en su conjunto. Por su parte, el segundo nivel, o **Nivel de Status**, está relacionado con todas aquellas nociones que para identificarse necesitan de la referencia a un status concreto (aunque genérico).

La arquitectura NÓESIS incluye también dos transformaciones, debiéndose considerar aquí el término *transformación* en un sentido amplio, como iremos mostrando a lo largo del presente trabajo según se analice la arquitectura desde uno u otro niveles de abstracción. La transformación denotada por T_0 captura la idea de que "existe una situación inicial". Lo representado en el Nivel de Independencia de Status refiere a un comportamiento genérico. Cuando dicho comportamiento se particulariza, es decir, cuando se representa una *ejecución* concreta de tal comportamiento, dicha ejecución concreta debe comenzar en una situación inicial, la cual pertenece al Nivel de Status. Por su parte, la transformación denotada por T recoge la idea de"paso de la situación actual a otra diferente". Cada una de las situaciones particulares caen dentro del Nivel de Status, y cada ejecución concreta consiste *a grosso modo* en una sucesión de situaciones particulares: la transformación T representa los cambios entre una situación y la siguiente en dicha sucesión.

En los siguientes sub-apartados vamos a detallar la interpretación de la arquitectura en función del nivel de abstracción que se considere en cada caso. Con *nivel de abstracción* nos estamos refiriendo aquí a los Niveles de Aplicación, Método y

Axiomático que han sido analizados en el Capítulo 1 del presente trabajo, y que establecen la distinción entre por ejemplo las nociones de modelo (de primer nivel), de lenguaje para desarrollar modelos, de metamodelo, etc. Aunque en la sección 3.1 hemos considerado la existencia de las vistas Estática y Dinámica para formalismos de comportamiento (es decir, al Nivel del Método), como veremos de inmediato esta separación es fácilmente detectable también, por ejemplo, en el Nivel de Aplicación. La arquitectura NÓESIS formaliza la existencia de estas vistas, de forma que se puede afirmar que el Nivel de Independencia de Status se corresponde con la Vista Estática y el Nivel de Status y las dos transformaciones se corresponden con la Vista Dinámica. Es necesario notar, sin embargo, que la transformación T_0 se encuentra de algún modo en el límite de la separación entre ambas vistas, ya que de hecho constituye la vía de comunicación entre ellas.

3.2.1. La arquitectura en el Nivel de Aplicación

Para explicar el significado de la arquitectura NÓESIS en el Nivel de Aplicación utilizaremos varios modelos de un mismo ejemplo. Inicialmente nos concentraremos en la explicación del significado de los dos niveles de la arquitectura, y posteriormente realizaremos un análisis similar para las dos transformaciones.

Supongamos que deseamos elaborar un modelo para reflejar parte del comportamiento de un termostato programable (este ejemplo es un pequeño fragmento del principal ejemplo de modelo dinámico que se considera en [RBP+91]). El termostato tiene como interfaz con el exterior dos elementos principales: un termómetro que le permite obtener la temperatura real y una pequeña consola a través de la cual el usuario puede fijar la temperatura a la que desea que se encuentre la habitación. El termostato mantendrá la caldera de la calefacción apagada mientras la temperatura exterior sea superior o igual a la temperatura programada, y mantendrá la caldera encendida cuando la temperatura exterior sea inferior a la programada. En esta descripción textual del dispositivo se está haciendo referencia a nociones relacionadas con el comportamiento del termostato pero que *son independientes de en qué situación concreta* se halle éste: la caldera puede estar encendida o apagada, hay una temperatura exterior y una programada, y ambas temperaturas deben compararse permanentemente. Es necesario resaltar como el propio lenguaje natural proporciona algunas de las claves para decidir que lo que estamos describiendo no

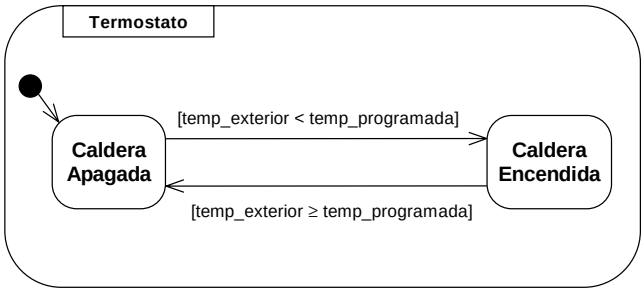


Figura 3.9: Statechart para el termostato

está ligado a ninguna situación concreta del sistema: la caldera *puede estar* encendida o apagada; las temperaturas *deben compararse*. Bajo la perspectiva de la arquitectura NÓESIS esta descripción se engloba por tanto en el Nivel de Independencia de Status. Sin embargo, si imaginamos el termostato en funcionamiento, aparecen nociones que sí están *relacionadas con cada situación particular*: la temperatura exterior *tendrá un valor*, la temperatura programada almacenará otro, una de las dos temperaturas *será mas alta* que la otra (o serán iguales), y en función de ello la caldera *se encontrará* o bien encendida o bien apagada. Así, la descripción de cualquier situación particular se hace en términos distintos a los utilizados cuando se describe el comportamiento de forma genérica, y por tanto la descripción de cada situación particular pertenece al Nivel de Status de la arquitectura NÓESIS.

Las descripciones anteriores, que constituyen un modelo textual, se encuentran al Nivel de Aplicación, y podemos observar una situación análoga si para desarrollar el modelo utilizamos una notación gráfica. En concreto, podemos describir un modelo statechart para representar el comportamiento del termostato: un posible modelo tal aparece en la figura 3.9.[2]

La distinción entre los niveles de la arquitectura ya se ha hecho notar en la

[2]Si se observa ahora de nuevo la figura 3.1 es fácil ver que el modelo statechart del termostato es de algún modo una concreción del modelo statechart del Sistema X. Esto es debido a que en la sección 3.1 hemos centrado la atención en la diferenciación entre lo que hemos llamado Vista Estática y Vista Dinámica *en lo relativo a los formalismos*, es decir, al Nivel del Método. Por ello, hemos creído conveniente en la sección anterior abstraer, en la medida de lo posible, los detalles sobre el modelo concreto que se utilizaba como ejemplo. Llegados al punto en que nos encontramos actualmente, es necesario observar, como se ha dicho algo más arriba, cómo se pueden considerar también una Vista Estática y una Vista Dinámica en el Nivel de Aplicación, y por tanto como la arquitectura se puede utilizar como marco genérico también en este nivel.

descripción textual del termostato. En cuanto al modelo gráfico (statechart en este caso), las ideas fundamentales expuestas en la sección 3.1 son también aplicables aquí. En particular, es evidente que lo que la figura 3.9 representa es el comportamiento genérico del termostato, y no es un tipo de diagrama habilitado para la representación de situaciones particulares como las descritas antes para el termostato: en el statechart no se representa explícitamente que, en un momento dado, una de las dos temperaturas será mas alta que la otra, o que la caldera se encontrará, por ejemplo, encendida. Desde la perspectiva de la arquitectura NÓESIS, la figura 3.9 –la cual pertenece al Nivel de Aplicación– es un modelo que cae dentro del Nivel de Independencia de Status. Como ya se ha hecho notar también en la sección 3.1, Statecharts no proporciona medios gráficos para representar situaciones particulares, y por tanto con la notación Statecharts estándar no es posible especificar modelos que representen tales situaciones, modelos que caerían dentro del Nivel de Status. En la sección 3.1 se ha mostrado una posible ampliación de la notación gráfica de Statecharts que permitiría representar este tipo de modelos, y en particular nos permitiría representar situaciones particulares del termostato. Como pequeña muestra de este hecho incluimos la figura 3.10. En él se ha utilizado la misma notación de la sección anterior, incluyendo además una especificación para el valor de 'VERDAD' o 'FALSO' de cada condición que constituye el disparador de las transiciones del statechart.[3]

Hasta ahora hemos analizado fundamentalmente el significado de los dos niveles de la arquitectura al Nivel de Aplicación. Corresponde ahora analizar como deben interpretarse las dos transformaciones de la arquitectura en este nivel de abstracción, y para ello continuaremos utilizando el ejemplo de aplicación particular que

[3]Con respecto a las notaciones Statecharts desarrolladas para especificar los modelos que pertenecen al Nivel de Status es necesario hacer una precisión referente a su ubicación dentro de los niveles de abstracción. Puesto que lo que se ha propuesto es una sintaxis concreta para algunos conceptos del formalismo Statecharts, dicha notación pertenece a la definición del propio lenguaje, y por tanto, si hubiéramos especificado con todo detalle la notación gráfica propuesta, nos encontraríamos en el Nivel del Método. Por el contrario, nos hemos limitado a mostrar ejemplos de tal posible notación, expresando diversos modelos statechart "ampliados", y estando situados, por tanto, en el Nivel de Aplicación. Observemos como sin embargo Redes de Petri es un formalismo que en su propia definición (Nivel del Método) incluye símbolos gráficos explícitos para representar situaciones particulares, concretamente a través de la noción (y la notación) de marca de un lugar. Incidiremos sobre ello en el siguiente apartado.

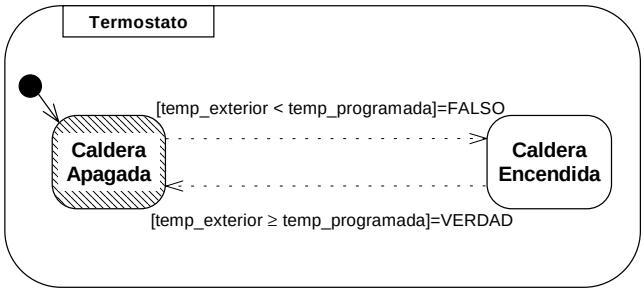


Figura 3.10: Statechart para el termostato en situación inicial

hemos presentado. Recordemos cómo, en la descripción textual del termostato, en el momento en que se establecía la diferencia entre la información relativa a uno y otro niveles de la arquitectura decíamos literalmente: "si imaginamos el termostato en funcionamiento...". El hecho de considerar el funcionamiento del termostato (obviamente imprescindible si lo que estamos analizando es el propio comportamiento) nos lleva de manera automática a considerar que en un momento dado el termostato *se encenderá*, y de forma inmediata se encontrará en la situación inicial. El establecimiento de cuál es esa situación inicial lleva consigo una serie de consideraciones. En el momento del encendido se podría por ejemplo considerar que el termómetro que recoge la temperatura exterior proporcione dicha temperatura, y que la caldera esté en posición de apagado (por ejemplo por motivos de seguridad). Sin embargo, ¿cuál debería ser la temperatura programada en la situación inicial? ¿Debería existir un valor por defecto para tal temperatura o debería aparecer algún tipo de alerta en el dispositivo que indique al usuario que debe fijar la temperatura deseada antes de que éste empiece a funcionar? No vamos a contestar a estas preguntas, que planteamos de modo retórico, ya que son cuestiones que tienen que ver con el diseño particular de cada dispositivo, y cada diseñador (o cada fabricante) podrá elegir la que sea más adecuada a sus intereses. Lo que nos interesa es hacer notar que en efecto existen algunas características del termostato que es necesario analizar y valorar cuando de lo que se trata es de determinar la situación inicial. La transformación T_0 de la arquitectura es la que refleja esta situación: dado un modelo genérico de comportamiento, en dicho modelo existirán diversos elementos para los que habrá que fijar sus "valores" iniciales. En el Nivel de Aplicación, una hipotética especificación de la transformación T_0 fijaría en una primera aproximación *cuáles* son los elementos del modelo para los que es necesario determinar su valor inicial. En el ejemplo del

termostato que nos ocupa, tres serían estos elementos: las dos temperaturas y el estado de la caldera. Una aproximación más detallada podría eventualmente establecer *cómo* se deben obtener dichos valores iniciales: en el termostato, podríamos fijar que en efecto la temperatura exterior es inmediatamente proporcionada por el termómetro (pero igualmente válido sería un diseño en el que esta temperatura tuviera un valor inicial "fijado de fábrica", y tras el encendido debiera considerarse un fase previa de "puesta a punto", hasta que el termómetro proporcionara la temperatura real) o que inicialmente la caldera está apagada, tal y como observábamos anteriormente. Retomemos ahora el modelo gráfico que hemos especificado para el termostato (statechart de la figura 3.9). Allí se observa que la necesidad de fijar valores iniciales es tenida en cuenta, mas tan sólo en parte, por la notación gráfica de Statecharts. En efecto, el símbolo que consiste en un punto relleno del que sale una flecha se corresponde con el concepto *transición por defecto*, que, aunque de forma genérica se utiliza para fijar el estado por defecto de entre los hijos de un estado OR, sirve en particular para determinar el estado activo inicial del statechart, cuando el estado OR es en concreto el estado raíz (es necesario hacer notar que al presentar esta descripción del significado del símbolo en cuestión estamos en realidad subiendo al siguiente nivel de abstracción, al Nivel del Método, pero se ha incluido aquí para facilitar la comprensión de la discusión que estamos efectuando sobre el Nivel de Aplicación). De esta forma, la observación del diagrama 3.9 nos proporciona no sólo información referente al comportamiento genérico del termostato (es decir, referente al Nivel de Independencia de Status), sino que nos proporciona también parte de la información que en nuestra arquitectura hemos ligado a la transformación T_0, en concreto el hecho de que en la situación inicial, el termostato debe mantener la caldera apagada. Hagamos notar de manera explícita que por el contrario el statechart 3.9 no proporciona información de ningún tipo acerca de cuáles deberían ser los valores iniciales de las temperaturas exterior y programada.

Para acabar con el análisis del significado de la arquitectura NÓESIS en el Nivel de Aplicación, vamos a considerar en este nivel la transformación T. Esta transformación es la que se corresponde de manera más directa a lo que nos referimos, en general de modo informal, cuando hablamos del "comportamiento del sistema", es decir, qué es lo ocurre, qué acontecimientos van sucediendo conforme el sistema evoluciona (como se puede deducir del análisis realizado hasta este punto, es evidente que "comportamiento" es mucho más que exclusivamente "evolución del sistema",

aunque es frecuente abusar del lenguaje en estos términos). Recordemos que el Nivel de Status recoge las diferentes situaciones particulares del sistema: la transformación T, que sale y llega de este nivel, representa el paso de una situación particular a la siguiente. En el caso de la descripción textual del termostato, la práctica totalidad de esta descripción contiene información relativa a la transformación T (recordemos que se decía "el termostato mantendrá la caldera de la calefacción apagada mientras la temperatura exterior sea superior o igual a la temperatura programada, y mantendrá la caldera encendida cuando la temperatura exterior sea inferior a la programada"). Esto es debido principalmente a dos razones. En primer lugar en efecto la transformación T es la más ligada al "comportamiento" en sentido estricto, y en segundo lugar la utilización del lenguaje natural en el modelo textual que hemos presentado hace que la descripción sea necesariamente informal y algo confusa, ya que en esta descripción, como hemos estado analizando, las ideas que pertenecen a los distintos constituyentes de la arquitectura aparecen entremezcladas. Observemos cómo de hecho el modelo textual no ofrece información *explícita* sobre el cambio de situación particular, sino que básicamente define cuáles son las situaciones posibles. Aun así, la lectura de la descripción del termostato conlleva una interpretación *implícita* que constituye la definición de los cambios entre tales situaciones. El modelo statechart del termostato es algo más preciso en este sentido, pues la noción de *transición* ayuda a percibir la idea de cambio. Sin embargo, y puesto que en efecto un statechart estándar no captura explícitamente situaciones particulares, más difícilmente aún puede capturar el cambio entre tales situaciones. La noción de cambio en este caso está ligada a la noción de comportamiento de un statechart, cuestión que debe ser analizada desde el Nivel del Método.

3.2.2. La arquitectura en el Nivel del Método

La arquitectura en el Nivel del Método sirve para analizar las características de comportamiento de los formalismos. Aunque *a priori* se podría intentar analizar el significado de la arquitectura para cualquier técnica de modelización, el interés principal de su utilización estriba en el análisis de formalismos que hayan sido creados con el propósito fundamental de representar comportamiento. Al igual que en el apartado anterior, nos guiaremos por algunos ejemplos para tratar de explicar las distintas componentes de la arquitectura NÓESIS a este nivel.

En particular, volvamos a considerar el ejemplo principal del presente trabajo, el formalismo Statecharts. La mayor parte de las consideraciones que es necesario realizar acerca del significado de la arquitectura en este caso ya han sido expuestas en la sección 3.1, y además en el metamodelo de Statecharts que presentaremos incidiremos con mucho detalle en estas cuestiones, por lo que en esta sección vamos sólo a recordarlas de manera breve. De forma general, la arquitectura está relacionada con el comportamiento de los modelos que se pueden crear con el lenguaje Statecharts, es decir, con el comportamiento de los statecharts. Así el lenguaje proporciona una larga serie de nociones (*estado*, *estado-OR*, *estado-AND*, *transición*, *conector por defecto*, *conector historia*, etc.) que permiten crear modelos (visuales, pues cada uno de dichos conceptos tiene asociada una sintaxis gráfica concreta). Estos modelos, como hemos visto en el ejemplo anterior, pertenecen al Nivel de Independencia de Status, ya que representan el comportamiento genérico del sistema modelado. Por lo tanto, todos los conceptos que se utilizan para crear un statechart (como los citados antes) también pertenecen al Nivel de Independencia de Status cuando nos encontramos al Nivel del Método y lo que estamos describiendo es el lenguaje en sí.[4] En efecto, los conceptos citados sirven para especificar modelos que no hacen referencia a status concretos, y por tanto son conceptos que se utilizan *con independencia de cuál sea el status concreto* en un momento determinado. Por su parte, cuando se analiza el comportamiento dinámico de un modelo statechart, es necesario hacer referencia a otros conceptos, o para ser más precisos, a modificaciones, calificaciones o valoraciones de los conceptos que hemos encuadrado en el Nivel de Independencia de Status. La siguiente cita, extraída literalmente de [HN96, pág. 298], aclara nuestra afirmación:

> "El comportamiento de un sistema (...) es un conjunto de posibles ejecuciones (...). Una ejecución consiste en una serie de fotogramas detallados de la situación del sistema; a cada fotograma se le llama un **status**. El primero en la secuencia es el status inicial, y cada uno de los siguientes es obtenido a partir de su predecesor ejecutando un **paso**.(...)
> Un status contiene información sobre estados y actividades activas, valores de data-items y condiciones, eventos generados y acciones progra-

[4] Como se verá a lo largo del trabajo, esta afirmación deberá ser matizada, especialmente en lo referente al concepto *transición*, que pasa por ser uno de los más, si no el más, complejo de analizar para el formalismo Statecharts.

madas, y alguna información acerca de la historia del sistema (su comportamiento pasado)".

Es necesario precisar que la definición de comportamiento que aparece en la cita se corresponde con la de una herramienta CASE particular, STATEMATE, pero puede ser considerada como una ampliamente admitida definición elemental de comportamiento de un statechart. De hecho, observemos que lo que se pretende definir es el "comportamiento de un sistema", pero en realidad este comportamiento se define en términos de status, que según la propia definición, es un concepto que refiere de manera directa y exclusiva a nociones Statecharts. De ello se deduce que el pretendido "comportamiento del sistema" en esta cita está haciendo referencia en realidad al comportamiento de un modelo (statechart).

En efecto pues, existen una serie de nociones Statecharts (tales como *estado activo*, *transición disparable*, *valor de variable*, *valor de condición*, *configuración básica*, etc.) que se utilizan al tratar con status concretos y deben encuadrarse en el Nivel de Status de la arquitectura NÓESIS.

Con respecto a las transformaciones, ya hemos comentado que el concepto *conector por defecto* tiene un curioso rol (por lo excepcional en el conjunto de conceptos Statecharts) que sí que está relacionado con el establecimiento del status inicial, es decir, con aquella información ligada a la transformación T_0. Sobre el resto de conceptos para los que es necesario proporcionar valores iniciales (particularmente variables y condiciones primitivas) la definición del lenguaje Statecharts no observa ya no cómo deberían obtenerse tales valores, sino tan siquiera una especificación detallada de qué conceptos necesitan de tal asignación. Por su parte, la transformación T para el caso de Statecharts se corresponde aproximadamente con el concepto de *paso*, definido de manera somera en la cita anterior de [HN96]. En esta misma referencia se puede encontrar (pág. 312 y siguientes) una descripción más detallada de en qué consiste un paso, a través de un algoritmo escrito en un pseudocódigo *ad hoc* (pues contiene importantes fragmentos en lenguaje natural). La arquitectura NÓESIS permite la representación del concepto de paso de manera más formal, y como veremos más adelante, también más comprensible gracias a la incorporación de una noción de refinamiento de las transformaciones de la arquitectura.

A lo largo de la presente sección estamos mostrando cómo en efecto la arquitectura NÓESIS se puede utilizar en diferentes niveles de abstracción. Pero también se ha observado al principio de la sección que además la arquitectura sirve como

marco general para el análisis de características de comportamiento dentro del Nivel del Método, es decir, con independencia de cuál sea el formalismo concreto de comportamiento que se esté considerando. Dicho de otro modo, la arquitectura no se ha diseñado de forma que esté asociada a un lenguaje particular, sino que se puede utilizar para analizar distintos formalismos de comportamiento. Con el propósito de ilustrar este hecho, vamos a mostrar cuál sería la interpretación de la arquitectura NÓESIS para Redes de Petri [Mur89, Pet81].

Existen numerosas formas alternativas de definir redes de Petri, y aquí optaremos por una definición simplificada de la presentada en [Pet81]. El objetivo fundamental es que el ejemplo sea fácilmente comprensible, y por tanto consideraremos el tipo de red de Petri más sencillo posible, eliminando explícitamente redes de Petri con múltiples arcos entre un lugar y una transición, redes de Petri con pesos, redes de Petri coloreadas o cualquier otro tipo de variante. Así, definiremos la *estructura de una red de Petri* como una 4-tupla (L,T,E,S) formada por un conjunto finito de *lugares* L, un conjunto finito de *transiciones* T y dos funciones, una *función de entrada* E y una *función de salida* S, que asocian, cada una de ellas, a cada transición un conjunto de lugares. Esta definición formal es acompañada de forma prácticamente automática por la noción de *grafo de una red de Petri*, que permite proporcionar una notación gráfica a cada red de Petri. En un grafo de red de Petri, existen dos tipos de nodos: cada lugar se representa por un círculo y cada transición por una barra vertical. De hecho, en Redes de Petri es habitual fusionar el concepto con la representación gráfica del mismo, y así se dice que los círculos *son* lugares y las barras verticales *son* transiciones. Para cada transición t de la red, se dibuja en el grafo un arco dirigido entre dicha transición y cada lugar que aparezca en el conjunto imagen de la función de salida aplicada sobre t (a los lugares implicados se les llama comúnmente *lugares de salida* de t). Análogamente, se dibuja un arco dirigido entre cada lugar y t, para cada lugar que aparezca en el conjunto imagen de la función de entrada aplicada sobre t (a tales lugares se les denomina *lugares de entrada* de t). Por ejemplo, el grafo de red de Petri de la figura 3.5 se correspondería con la siguiente estructura de red de Petri:

$$L = \{A, B, e, f\}$$
$$T = \{t_1, t_2\}$$
$$E(t_1) = \{A, e\} \quad S(t_1) = \{B\}$$
$$E(t_2) = \{B, f\} \quad S(t_2) = \{A\}$$

Tanto la noción global de estructura de red de Petri (o su representación gráfica asociada) como los conceptos concretos de lugar, transición y funciones de entrada y salida (o respectivamente los círculos, transiciones y arcos) son mecanismos que se definen en el formalismo con la intención de representar algo genérico, es decir, no ligado a un estado (status) concreto del diagrama red de Petri. Así, la estructura de una red de Petri (o, insistimos, su grafo asociado) caen dentro del Nivel de Independencia de Status de la arquitectura NÓESIS[5].

Una vez definido lo que es la estructura de una red de Petri, el siguiente concepto que es necesario definir es el de *marcado de una red de Petri*. Un marcado μ es la asignación de un cierto número entero no negativo de marcas a cada lugar de una estructura de red de Petri. Desde el punto de vista formal, este marcado puede definirse como una función del conjunto del lugares L en el conjunto de los números enteros no negativos, o en el caso de considerar este conjunto L como conjunto ordenado, como un vector de tantas componentes como elementos de L. Una *red de Petri marcada* es así una estructura de red de Petri junto con un marcado de ella, o bien una 5-tupla (L,T,E,S, μ). Para un grafo de red de Petri, un marcado se representa de manera gráfica colocando tantos puntos dentro de cada círculo que representa un lugar como marcas correspondan a dicho lugar. Así, la función μ para el grafo marcado de la figura 3.6 sería $(1, 0, 0, 0)$ (recordemos que para este ejemplo habíamos establecido que $L = \{A, B, e, f\}$), y para el grafo de la figura 3.7 sería $(1, 0, 1, 0)$. Con la introducción de la nociones de *marca*, *marcado*, *red de Petri* (o equivalentemente *grafo de red de Petri*) *marcada*, se contempla la noción de estado (status) de una red de Petri. De hecho, en el formalismo se define de forma explícita la noción de *estado de una red de Petri*, que consiste simplemente en su marcado. Una misma (estructura de) red de Petri puede encontrarse en diferentes situaciones en función de cuál sea su marcado en un momento dado. Por tanto, los conceptos de *marca*, etc., pertenecen al Nivel de Status de la arquitectura NÓESIS para el

[5]Observemos que por primera vez, y debido a que el propio formalismo Redes de Petri es definido en estos términos, se relaciona el término 'estructura' con el Nivel de Independencia de Status. Hasta ahora hemos tratado de evitar este paralelismo para no causar confusión con respecto a la diferencia global entre 'estructura' y 'comportamiento' tal y como es habitual en las metodologías de análisis y diseño orientadas a objeto. Sin embargo, sí es cierto que lo que el Nivel de Independencia de Status captura es aquello que podría denominarse 'parte estructural del comportamiento', aunque con el fin de facilitar la lectura del trabajo no utilizaremos este tipo de referencia.

formalismo Redes de Petri.

Finalmente, para entender el comportamiento básico de una red de Petri, es necesario describir en que consiste la *ejecución de una red de Petri*. Básicamente una red de Petri se ejecuta mediante el *disparo de transiciones*. El proceso de disparo de una transición conlleva la eliminación de marcas de sus lugares de entrada y la creación de marcas en sus lugares de salida (aunque parezca obvio, es conveniente hacer notar que de esta definición se deduce que la ejecución de una red de Petri sólo tiene sentido para redes de Petri marcadas, es decir, en terminología NÓESIS, para redes que representen un status). Una transición es *disparable* si cada uno de sus lugares de entrada está marcado[6]. En la figura 3.7, la transición t_1 es disparable, pues existe una marca en cada uno de sus lugares de entrada. La ejecución de dicha transición nos llevaría a la red de Petri con marcado $(0, 1, 0, 0)$, es decir, con una única marca en el lugar B. Por tanto, el disparo de una transición conlleva el paso de un marcado (un estado de la red) a otro. Si trasladamos la situación a la arquitectura NÓESIS, es claro que la transformación T se corresponde con el mecanismo de disparo de transiciones para redes de Petri. Existen otros muchos conceptos más complejos que caerían dentro de la transformación T, tales como la resolución de conflictos (qué ocurre cuando existen simultáneamente varias transiciones disparables) o la alcanzabilidad (si es o no posible obtener un cierto marcado a partir de otro), pero sobrepasan la profundidad con la que estamos abordando el ejemplo. Sí que es necesario hacer una observación acerca del sentido de la transformación T_0, que no es otro que el de la necesidad de fijar el *marcado inicial de una red de Petri* para poder considerar su ejecución. Esta necesidad sí que es observada por las definiciones estándar de redes de Petri, aunque su interpretación difiere de unos autores a otros. Peterson en [Pet81], referencia que hemos utilizado para desarrollar nuestro ejemplo, define de manera estrictamente separada lo que es la estructura de una red de Petri de lo que es el marcado de la red, de forma que el marcado inicial es para este autor un caso especial de marcado, que es el que hay que proporcionar de forma adicional (aunque insistimos, separada) para poder considerar la ejecución de la

[6]En este momento es cuando aparece la principal diferencia de nuestra definición de red de Petri con respecto a la que aparece en [Pet81], que es más general. En ésta última se permiten múltiples arcos entre un lugar y una transición fijadas –lo que implica por ejemplo que los conjuntos imagen de las funciones de entrada y salida ya no sean en general tales conjuntos simples, sino conjuntos con repetición–, y por tanto una transición será disparable si en cada uno de sus lugares de entrada hay al menos tantas marcas como arcos vayan desde el lugar hasta la transición.

red. Sin embargo, Murata en [Mur89] da una definición de red de Petri que contiene en sí el marcado inicial. Aunque obviamente necesita especificar por separado el significado de la estructura y el marcado, este autor no considera como red de Petri un grafo no marcado, sino que una red de Petri es siempre un grafo con su marcado inicial. Esta aparente contradicción muestra un ejemplo de cómo en efecto la tarea analizar las características que recoge la arquitectura NÓESIS para los formalismos de comportamiento no es ni mucho menos sencilla. Bajo nuestro punto de vista, la arquitectura NÓESIS puede utilizarse para mejorar la especificación y definición de los formalismos, haciéndolos más fácilmente utilizables y comprensibles para analistas, diseñadores e ingenieros del método.

3.2.3. La arquitectura en el Nivel Axiomático

Una vez analizada la interpretación de la arquitectura NÓESIS en los niveles de abstracción de Aplicación y Método, veamos cuál es su significado desde el punto de vista del nivel más abstracto, el Nivel Axiomático. Tal y como se ha analizado en el Capítulo 1, el Nivel Axiomático trata sobre aquellos artefactos que a su vez producen los artefactos que son propios del Nivel del Método. Por lo tanto, la especificación de una técnica de metamodelización concreta pertenece a este nivel, puesto que los productos de una técnica de metamodelización (metamodelos) son artefactos del Nivel del Método (formalismos o lenguajes de modelización).

El principal problema con que nos enfrentamos a la hora de interpretar la arquitectura NÓESIS desde el Nivel Axiomático es la pérdida de intuición respecto al significado de la palabra *comportamiento*. Es relativamente fácil tener una idea intuitiva de lo que significa el "comportamiento de un sistema", es decir, del comportamiento al Nivel de Aplicación. Como estamos viendo en este capítulo una idea intuitiva no es una idea completa, puesto que es necesario un análisis minucioso del significado del término, pero es cierto que es muy difícil hacer referencia a este tipo de nociones si no se tiene detrás esa idea intuitiva. Cuando "subimos" al Nivel del Método, la intuición todavía nos guía para indicarnos que el propósito de todo formalismo de comportamiento es en efecto el de "representar comportamiento de sistemas", y que por tanto se definirán los mecanismos y procesos adecuados en cada formalismo que permitan realizar esa representación, lo que sin duda llevará a la definición, al menos implícita, del "comportamiento del formalismo". Este

"comportamiento del formalismo" debe ser siempre entendido, y este es un punto fundamental, como el comportamiento de los *modelos* (modelos, que como tales, pertenecen al Nivel de Aplicación) creados con el formalismo. Al pasar del Nivel del Método al Nivel Axiomático, se reproduce en parte el patrón que hemos identificado al pasar del Nivel de Aplicación al Nivel del Método. En el Nivel Axiomático, sucede que *una (toda) técnica de metamodelización* debe ser capaz de "representar comportamiento de formalismos", es decir, *debe proporcionar mecanismos y procesos adecuados que permitan representar el comportamiento de los modelos* que se crean con los formalismos para los que eventualmente se creará un metamodelo con la técnica en cuestión. Dicho de otro modo: así como un formalismo de comportamiento debe ser capaz de representar comportamiento de sistemas (lo que se consigue mediante el suministro de las herramientas de modelización adecuadas y la consiguiente elaboración de modelos), una técnica de metamodelización debe ser capaz de representar comportamiento de modelos (lo que debe conseguirse mediante el suministro de las herramientas de (meta)modelización adecuadas y la consiguiente elaboración de (meta)modelos).

Por lo tanto, la arquitectura NÓESIS en el Nivel Axiomático debe entenderse como la descripción de aquellas herramientas que cada técnica de metamodelización proporcione para analizar el comportamiento de modelos, y en particular, cuáles de esas herramientas se utilizan para representar los elementos de modelización que cada formalismo proporcione, en función de si estos elementos se utilizan para modelar aquellas características de los modelos que son independientes del status concreto, aquellas que dependen de cuál sea el status, aquellas que sirven para fijar el status inicial o aquellas que se utilizan para modelar el cambio de status. De forma más concreta, el hecho de recoger la arquitectura en la especificación de una técnica de metamodelización puede conllevar la definición de nuevas herramientas que permitan la representación de las características de comportamiento tal y como sugiere la arquitectura, y en particular puede traer una nueva definición para la noción de metamodelo que cada técnica proponga. Es en este sentido donde la arquitectura aporta un nuevo punto de vista a las técnicas de metamodelización existentes. La mayoría de ellas (véanse, por ejemplo, [HSB97, KLR96, MBJK90, Sae95, VtH95]) concentra sus esfuerzos en la especificación de metamodelos de formalismos estructurales (es decir, formalismos con los que se pueden crear modelos que representan la parte estructural de los sistemas), y en el caso de presentar metamodelos de for-

malismos de comportamiento, se centran exclusivamente en los aspectos que dentro de la arquitectura NÓESIS se englobarían en el Nivel de Independencia de Status, y no analizan el resto de características relacionadas con el comportamiento que estamos presentando en este capítulo. En un próxima sección mostraremos como la arquitectura NÓESIS se puede incorporar a la técnica de metamodelización NÓESIS que hemos mostrado en el capítulo anterior, lo que dará lugar a una nueva definición de metamodelo NÓESIS. Sin embargo, del mismo modo que ocurría en el Nivel del Método, la arquitectura no ha sido pensada para estar ligada a una técnica de metamodelización particular, ni siquiera a la técnica NÓESIS, sino que puede utilizarse como marco general con independencia de cuál sea la técnica de metamodelización utilizada, aunque, como hemos observado algo más arriba, la adopción de la arquitectura por una técnica de metamodelización concreta puede llevar (como en el caso de la técnica NÓESIS) a una redefinición de la noción de metamodelo que dicha técnica proponga.

3.2.4. Refinamiento de la arquitectura Nóesis

Al principio de la sección actual hemos presentado una primera aproximación a la arquitectura NÓESIS, que hemos ido analizando, a lo largo de los últimos apartados, en función del nivel de abstracción concreto que fuera considerado en cada caso. Una conclusión elemental de todo el análisis efectuado hasta el momento consiste en señalar que el estudio pormenorizado de las características de comportamiento es arduo y complejo. Esta situación nos ha hecho considerar la posibilidad de que en la arquitectura NÓESIS pudieran observarse diferentes grados de detalle, lo que permitiría, en cierto sentido, un análisis *top–down* del comportamiento. Con la presentación actual de la arquitectura, se define un marco genérico y bastante abstracto para poder analizar las características de comportamiento. Un *refinamiento* de la arquitectura permitiría distinguir mayores niveles de detalle en tales análisis. De alguna forma lo que estamos haciendo es adaptar a nuestra perspectiva la dimensión de *granularidad*, la cual se considera una propiedad deseable para los formalismos de modelización. Así por ejemplo, en [RSM95] se afirma que «un formalismo de modelización de procesos sencillos debería acomodarse a un amplio rango de granularidad de modelo de una forma homogénea». Aunque la cita trata de un caso algo más particular (modelización de procesos), es evidente que una característica que

cualquier buen formalismo de modelización debería considerar es la posibilidad de construcción de modelos inicialmente "de grano grueso" (es decir, poco detallados, aunque útiles para obtener una visión general de conjunto) para después permitir el refinamiento de tales modelos, hasta conseguirlos con el nivel de detalle adecuado a cada necesidad particular.

A priori, y puesto que la arquitectura NÓESIS se compone de dos niveles y de dos transformaciones, un refinamiento de la misma podría abordarse desde cualquiera de las dos (o incluso simultáneamente las dos) posturas: posibilitar un refinamiento de los niveles o posibilitar un refinamiento de las transformaciones. Un refinamiento de los niveles debería consistir, básicamente en el establecimiento de subniveles. De forma intuitiva parece razonable pensar que lo que tales subniveles deberían representar serían aspectos diferenciados y más detallados con respecto a sus niveles "padre". Así por ejemplo, para el Nivel de Independencia de Status, podrían establecerse subniveles para capturar, por un lado aquellos elementos para los que es necesario proporcionar valores iniciales y por otro lado aquellos elementos para los que no, lo que haría, en particular, más sencilla la especificación de la transformación T_0. Entendemos, sin embargo, que el mayor nivel de complejidad dentro de la arquitectura se alcanza, en general, en las transformaciones, y más en concreto, en la transformación T. Si bien es cierto que cuánto más complejo sea el comportamiento a modelar, o más complejo sea el formalismo de comportamiento que se considere, más difícil será la especificación de la información relativa a los niveles de la arquitectura, aún más difícil será la especificación de las transformaciones, sobre todo, insistimos, la especificación de T. En función del nivel de abstracción que se analice, para cada sistema, para cada formalismo o para cada técnica de metamodelización, se deberá fijar cuál es el nivel de detalle adecuado para la especificación relativa a los niveles, y cómo se debe organizar la información respectiva, pero creemos que una excesiva concreción de los niveles de la arquitectura en este sentido podría limitar el grado de libertad que debe otorgarse a la tarea de modelización. A pesar de ello, como veremos de inmediato, el tipo de refinamiento que proponemos se encuentra íntimamente ligado a la información recogida en los niveles.

En concreto, el refinamiento que proponemos está orientado al aumento del nivel de detalle para las transformaciones. En el resto de la explicación vamos a concentrarnos en la transformación T, que como hemos observado es en general la más compleja de las dos. La idea básica es que la transformación puede dividirse en va-

rias transformaciones de menor complejidad. Para ello, es necesaria la introducción de la noción de fase intermedia en la especificación de la transformación original, de forma que esa especificación de T se pueda realizar en pasos más pequeños. A cada una de estas fases intermedias la denominaremos *fase de status virtual*. El calificativo de "virtual" se utiliza con el sentido de ser algo de existencia aparente, no real, interpretando esta afirmación dentro del contexto en que nos encontramos. Un status (de un sistema, de un modelo) es una situación "estable", es decir, una situación en la que potencialmente se podría residir de manera permanente ante la ausencia de nuevos estímulos en el sistema o modelo. Un *status virtual* es una situación que no tiene el rango de status, y en particular no es estable, pero que puede ser útil para describir el cambio entre un status y el siguiente (que es precisamente el objetivo primordial de la transformación T). A lo largo de esta sección ya nos hemos encontrado con situaciones que se encuadran fácilmente en este marco. Por ejemplo, la red de Petri de la figura 3.6 representa un marcado particular de la estructura de red de la figura 3.5, y constituye un status de la misma, pues de las reglas de ejecución del formalismo redes de Petri (que forman parte de la transformación T para este formalismo) se deduce que ninguna de las dos transiciones de la red son disparables, y por tanto la red no puede avanzar a un nuevo marcado. Sin embargo, la red de la figura 3.7 no representa un status de la red, ya que las mismas reglas de ejecución hacen que, puesto que la transición t_1 es disparable, y es la única en tal situación, debe dispararse, y por tanto el sistema no puede permanecer en la situación de la figura 3.7, red que por tanto debe considerarse un *status virtual*.

Así pues, el refinamiento que proponemos consiste en la división de la transformación T en una cadena de fases de status virtual y una serie de transformaciones (menos complejas que la original) entre cada par de estas fases. De esta forma, a partir de un status, se determinaría un status virtual a través de una transformación más sencilla. A partir de este status virtual se obtendría otro de un tipo diferente, a través de otra transformación, y así sucesivamente, hasta determinar el siguiente status, que se habría obtenido de forma progresiva. La figura 3.11 muestra una representación gráfica del refinamiento propuesto para la transformación T de la arquitectura.

La arquitectura no fija cuál debe ser el número concreto de fases de status virtual, ni por tanto, cuál es el número de transformaciones intermedias, que se deben especificar. El analista o meta–analista deberá fijar cuál es el número más adecuado

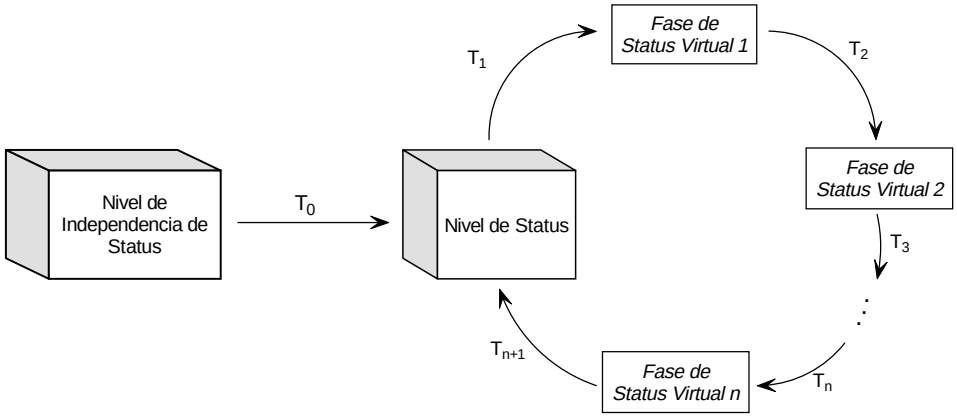


Figura 3.11: Refinamiento de la transformación T

para cada caso particular, siempre teniendo presente que cuantas más fases intermedias se introduzcan, el nivel de detalle con que se presentará la transformación será mayor, aunque, obviamente, este mayor nivel de detalle conllevará un mayor trabajo de especificación y modelización. En cualquier caso, un requisito imprescindible que debe verificarse para asegurar que el refinamiento efectuado es coherente con la situación de partida es que la transformación compuesta $T_{n+1} \circ T_n \circ ... \circ T_2 \circ T_1$ debe dar como resultado la propia transformación T.

3.2.5. Ventajas de la utilización de la arquitectura Nóesis

Aunque a lo largo de esta sección hemos ido mostrando algunas de las ventajas que proporciona el uso de la arquitectura Nóesis, parece adecuado presentarlas de manera conjunta, añadiendo algunas adicionales así como otros comentarios.

En primer lugar, la arquitectura Nóesis proporciona una interpretación de las características de comportamiento que puede ser usada de forma útil durante el desarrollo de sistemas en diferentes niveles de abstracción, en función de los diferentes puntos de vista que diferentes usuarios (tales como ingenieros del software, desarrolladores de herramientas, ingenieros del método, etc.) tengan sobre el tema. Por ejemplo, un analista o diseñador de sistemas usará este enfoque de comprensión del comportamiento para obtener una interpretación más precisa del sistema que esté siendo modelizado. En particular, esta tarea de modelización será más fácil si el lenguaje utilizado por el ingeniero del software para modelizar las características de comportamiento ha sido descrito de forma precisa siguiendo las directrices de la

arquitectura NÓESIS.

A su vez, estos lenguajes para la modelización de comportamiento pueden formar parte de herramientas CASE, herramientas que sin duda son de gran utilidad para el desarrollo de software. La consideración de nuestra arquitectura puede proporcionar soporte para el análisis de herramientas CASE existentes y para el desarrollo de otras nuevas, siempre en función de las intenciones de los desarrolladores de estos programas. Si fijamos nuestra atención una vez más en Statecharts, tal y como ya ha sido dicho, un statechart estándar se corresponde con el Nivel Independiente de Status de la arquitectura. Por tanto, una herramienta exclusivamente de dibujo de statecharts puede concentrar sus esfuerzos en este Nivel. Sin embargo, un desarrollador de CASE que esté interesado en dotar a su herramienta de características de animación o de simulación deberá tener en cuenta que la ejecución simulada de un modelo statechart involucra la aplicación combinada de los conceptos relativos a la transformación T_0, el Nivel de Status, y la transformación T, aplicados al caso particular del formalismo Statecharts. Algunas herramientas CASE recientes que integran el uso de UML State Machines se basan en enfoques similares, ya que están dotadas de características de animación. Por ejemplo, la herramienta Rhapsody[7] proporciona vistas animadas de la aplicación que se está modelizando, y en particular permite la observación del comportamiento de un state machine utilizando un esquema de colores para diferenciar, por ejemplo, entre estados activos y no activos. Sin embargo, debemos insistir en que este tipo de enfoques pertenecen a herramientas concretas, y no se basan en ninguna definición estándar ni de Statecharts ni de State Machines. En cualquier caso, un analista o diseñador de sistemas se beneficiaría del uso de una herramienta CASE que siguiera la arquitectura NÓESIS, ya que tal herramienta proporcionaría como sub–producto un patrón para la modelización de comportamiento. En particular, la herramienta no debe dificultar el trabajo del ingeniero del software por ejemplo mediante una especificación detallada de las transformaciones de la arquitectura, sino que basta que con que le informe de la disponibilidad de estas utilidades dentro de la herramienta.

Por último destaquemos una vez más que una de las principales utilidades de la arquitectura NÓESIS es su uso como artefacto de la ingeniería del método. En particular, si se usa la arquitectura, con el nivel de detalle que sea necesario en cada caso, en distintos métodos o versiones de métodos, es más sencillo establecer un

[7] http://www.ilogix.com/rhapsody/

marco para la comparación de estos métodos. Por ejemplo los métodos derivados de otros pueden ser comparados con los originales. Como curiosidad, y puesto que Statecharts y Redes de Petri son los dos formalismos a los que estamos haciendo referencia constantemente en este capítulo, la arquitectura nos permitiría comparar el formalismo conocido como PetriCharts (que se obtiene mediante «una adopción natural de los principios de Statecharts en la teoría de Redes de Petri» [HV95]) con los formalismos de los cuales se deriva. Otro caso en el que la arquitectura podría ser útil es en el de la comparación de diferentes versiones de métodos. Por ejemplo, existen multitud de versiones distintas de Statecharts, tal y como se muestra en [vdB94], y la arquitectura puede facilitar la detección de sus diferencias. Un ejemplo concreto de esta situación se mostrará en el capítulo siguiente durante la presentación de un metamodelo NÓESIS de la versión de Statecharts que se muestra en [HN96, HP98].

3.3. La noción de metamodelo de comportamiento Nóesis

A lo largo de la sección anterior hemos presentado una arquitectura que permite representar características de comportamiento de forma abstracta y general. En particular, hemos mostrado que la arquitectura tiene una interpretación desde el Nivel Axiomático, que se traduce en el hecho de que un lenguaje de metamodelización debe disponer de conceptos adecuados que permitan la representación de formalismos de comportamiento, y en particular, que permitan representar los mecanismos precisos que cada uno de estos formalismos fija para expresar el comportamiento de sus modelos.

En este sentido, la aplicación de la arquitectura a un lenguaje de metamodelización puede conllevar una redefinición de la noción de metamodelo que tal lenguaje proponga. En este trabajo se propone una redefinición tal para el caso de la técnica NÓESIS. Esta técnica, en su forma elemental, tal y como se ha presentado en el capítulo anterior, permitía la representación del comportamiento de los modelos a través de complicadas restricciones expresadas básicamente en lenguaje natural (véase [DZR97]). Esta forma de expresar el comportamiento, si bien puede ser válida para modelos de comportamiento 'sencillo', no parece la más adecuada cuando

la dinámica del sistema representado tiene un alto grado de complejidad, ya que se dificulta la legibilidad y comprensibilidad del metamodelo. La redefinición que ahora proponemos sustituye la mayor parte de las restricciones por una aplicación de la arquitectura NÓESIS que hemos presentado en este capítulo.

De la misma forma que hemos denominado formalismos de comportamiento a aquellos formalismos que hayan sido diseñados para la representación de comportamiento de sistemas, llamaremos *metamodelo de comportamiento* a un metamodelo que represente un formalismo de comportamiento. Para el caso particular de la técnica de metamodelización NÓESIS, hemos de proporcionar una definición para la noción de *metamodelo de comportamiento* NÓESIS, que consiste en una ampliación de la noción básica de metamodelo NÓESIS, utilizando las ideas que aporta la arquitectura. Tal ampliación se realiza detallando algunas de las componentes que constituyen un metamodelo NÓESIS básico. En concreto, un metamodelo de comportamiento NÓESIS consta de:

- perspectiva
- sistema de referencia
- marco representacional, el cual contiene un conjunto de soportes *independientes del status* y un conjunto de restricciones, y
- definición de modelo, la cual contiene, a su vez
  - un conjunto de soportes *status*
  - una transformación T_0, que a partir de cualquier modelo del conjunto de soportes independiente del status, obtiene un modelo del conjunto de soportes de status
  - una transformación T, que a partir de cualquier modelo del conjunto de soportes status obtiene otro modelo de este mismo conjunto de soportes

Las diferencias con respecto a la definición previa de metamodelo NÓESIS son varias. Por una parte, el único conjunto de soportes que existía en la definición anterior pasa a ser el conjunto de soportes independientes del status, y se alinea así con el nivel correspondiente de la arquitectura. Por otra parte, la diferencia más importante radica en la componente de definición de modelo de cada metamodelo,

que conlleva la especificación de otro conjunto de soportes, que se corresponde con el Nivel de Status de la arquitectura NÓESIS, y dos transformaciones correspondientes con las dos transformaciones de la arquitectura. Recalquemos que la finalidad del segundo conjunto de soportes que debe definirse para proporcionar un metamodelo de comportamiento es muy diferente de la que tiene el hecho de definir un conjunto de soportes para determinar el marco representacional. En este último caso, el objetivo era el de permitir un cierto grado de modularidad en la especificación de los soportes, lo que además facilita la legibilidad de los mismos. En el caso del conjunto de soportes de status, si bien existirán conceptos que aparecerán también en el conjunto de soportes independientes de status, el objetivo fundamental de los soportes de status es el de representar la información ligada a la representación de status concretos, y en particular representar aquellos conceptos que son utilizados por cada formalismo de comportamiento para modelar ese tipo de información. Por su parte, la aparición de transformaciones como componentes integrantes del metamodelo también supone una novedad respecto a la definición inicial de metamodelo NÓESIS. Este hecho implica la necesidad de una noción básica de transformación dentro de la técnica de metamodelización NÓESIS, noción que se encuentra en desarrollo en el momento de escribir el presente trabajo. Algunas aplicaciones de una noción de transformación ya han sido utilizadas por miembros del Grupo NÓESIS en diferentes contextos tales como construcción y modificación de soportes [DZ], interoperabilidad de métodos [DZ00] o adaptación de métodos [DZE00]. Observemos, sin embargo, que el tipo de transformación necesaria para especificar la componente de definición de modelo tiene un carácter distinto al que se necesitan para las tareas citadas. Así, para modificar soportes o para permitir la interoperabilidad de métodos (que estarán representados a través de un metamodelo, y en particular a través de conjuntos de soportes) es necesario definir *transformaciones entre soportes*. Sin embargo, las transformaciones necesarias para la definición de modelos son, precisamente, *transformaciones entre modelos*, que si bien pueden guardar relación con las transformaciones entre soportes, no son en general transformaciones derivadas de éstas.

Estas ideas motivan la necesidad de fijar, de manera algo más precisa, la noción de modelo para un metamodelo de comportamiento NÓESIS, y cuál es su relación con la noción de modelo de metamodelo NÓESIS que habíamos definido en el capítulo anterior. Recordemos que, en general, un modelo de un metamodelo NÓESIS se

construía como un conjunto de instancias de los conceptos fuente de soporte, de tal forma que cada una de estas instancias quedaba descrita a partir de instancias de los conceptos atributivos propios y de instancias de los conceptos atributivos heredados para el concepto fuente de soporte. Estas instancias, además, debían cumplir las restricciones locales y globales que se hubieran especificado en el marco representacional. Podemos conservar esta idea, de forma que un modelo tal sea un modelo de un conjunto de soportes determinado. Así, podremos hablar de modelos del conjunto de soportes independientes del status y de modelos del conjunto de soportes de status. Dicho esto, un *modelo de un metamodelo de comportamiento* NÓESIS será un modelo del conjunto de soportes independientes del status, junto con la consideración de la aplicación, a dicho modelo, de las transformaciones especificadas en la componente de definición de modelo del metamodelo. La aplicación de las transformaciones conlleva la aparición de nuevos modelos. Así, dado el modelo inicial (que como acabamos de comentar será un modelo del conjunto de soportes independientes de status), a través de la transformación T_0 obtendremos un modelo del conjunto de soportes de status. Este modelo es una representación del status inicial del sistema. A su vez, sobre este modelo actuará la transformación T, de forma que se obtiene otro modelo del conjunto de soportes de status, representando el siguiente status en que se encuentre el sistema. Este esquema de actuación de la transformación T debe repetirse, de forma que se obtiene toda una serie ordenada de modelos del conjunto de soportes de status. Bajo nuestro punto de vista, la consecución de esta serie de modelos y los consiguientes pasos de uno a otro, es lo que hace que nuestro enfoque sea adecuado para representar las características de comportamiento desde una perspectiva de metamodelización.

En la noción de metamodelo NÓESIS que acabamos de presentar se han incorporado las ideas de la arquitectura en su forma más elemental, según hemos presentado en la sección 3.2. Sin embargo, hemos visto también que la arquitectura admite la posibilidad de introducir cierto grado de refinamiento, fundamentalmente en la especificación de las transformaciones, y más en particular de la transformación T. Si aplicamos este refinamiento concreto de la arquitectura a la definición de metamodelo NÓESIS, obtenemos a su vez una especificación más detallada de la componente de definición de modelo. En particular, por cada Fase de Status Virtual de la arquitectura que se considere, se debe especificar un *conjunto de soportes de status virtual*, lo que conlleva la definición de modelos de tal conjunto. Según marca la

arquitectura refinada, es necesario especificar además nuevas transformaciones entre modelos que permitan obtener los modelos de los conjuntos de soportes intermedios.

Observemos también que aunque a la nueva noción la hemos llamado metamodelo de comportamiento, esta noción es una generalización de la noción de metamodelo básica, y por tanto puede ser también utilizada para expresar metamodelos que representen formalismos que no sean de comportamiento, simplemente retornando a la componente de definición de modelo básica, y no especificando ni el conjunto de soportes de status ni las dos transformaciones. La denominación de conjunto de soportes independientes de status sigue siendo válida, puesto que si el formalismo es de tipo estructural, la información que debe representarse en los soportes ciertamente no depende de ningún status determinado.

Capítulo 4

Un Metamodelo Nóesis de Statecharts

En el capítulo anterior se ha presentado nuestra interpretación de la modelización de comportamiento, lo que nos ha llevado a proponer una arquitectura general para representar las características de comportamiento. Esta arquitectura ha sido utilizada para obtener una definición ampliada de metamodelo NÓESIS, de forma que un metamodelo tal pueda capturar de manera adecuada las características propias de los modelos que se especifiquen como instancias del metamodelo, y que serán, a su vez, modelos del formalismo (de comportamiento) que se esté (meta)modelizando.

El objetivo del presente capítulo es el de mostrar un ejemplo de la situación que se acaba de describir. En particular, se va a presentar un metamodelo NÓESIS para el formalismo Statecharts. Dos razones han sido las que han hecho que hayamos elegido Statecharts como caso de estudio. En primer lugar, el elevado grado de complejidad del formalismo. La elección de un lenguaje de comportamiento más sencillo o la consideración de un subconjunto del lenguaje Statecharts hubieran facilitado la lectura del trabajo, pero en tal caso no hubiéramos podido mostrar las distintas implicaciones y dificultades que conlleva la elaboración de un metamodelo 'real', es decir, de un metamodelo de un formalismo utilizable en la práctica. En segundo lugar, Statecharts es un lenguaje ampliamente utilizado, y sobre el que pueden encontrarse multitud de referencias en la literatura. Sólo a modo de ejemplo, y sin intención de ser exhaustivos, citaremos algunos de los trabajos más destacados publicados sobre Statecharts. Desde la presentación original en [Har87] del formalismo y de la primera semántica en [HPSS87], se han desarrollado multitud de variantes, como puede

verse en [vdB94], y se han escrito multitud de trabajos analizando diferentes características del lenguaje (véanse, por ejemplo, [HKCK95, MSPT96, EGKP97, Gei99]) o describiendo diferentes semánticas para Statecharts [HRdR92, HN96, LdBC00]. El desarrollo de este lenguaje también ha dado lugar a la creación de algunas herramientas CASE tales como STATEMATE, sobre la que también podemos encontrar diferentes publicaciones [HLN+90, HN96, HP98, FH]. Aunque originalmente Statecharts fue ideado como un lenguaje para la modelización de sistemas reactivos, recientemente se ha investigado en la conexión del formalismo con el paradigma de la orientación a objeto [HG97, MK98], y en concreto el uso del lenguaje ha experimentado un fuerte impulso desde que ha sido adoptado y adaptado por varias metodologías de modelización orientadas a objeto [RBP+91, SGW94]. En particular, la inclusión de una variante de Statecharts con el nombre de *State Machines* dentro del Lenguaje Unificado de Modelización (UML) ha provocado la aparición, una vez más, de multitud de trabajos relacionados con el tema: [LP99a, LP99b, Dou99, LMM99, DRZ00b, RKR+00, BCR00]. El éxito de UML y por tanto de UML State Machines nos han llevado a elaborar un metamodelo diferenciado para esta variante: este metamodelo, que mostraremos en el siguiente capítulo, demuestra además la flexibilidad de la arquitectura NÓESIS, que puede utilizarse con independencia tanto del lenguaje de metamodelización elegido, como del lenguaje o formalismo metamodelizado. Aún a pesar de que UML se ha convertido en un importantísimo foco de interés para la investigación, el interés del lenguaje Statecharts "estándar" en el momento presente queda demostrado por la aparición constante de nuevos artículos relativos al formalismo (véanse trabajos muy recientes como [GHD01, Sch01, CABJ01]).

4.1. Introducción al metamodelo

En los anteriores capítulos del presente trabajo hemos presentado algunas ideas básicas sobre metamodelización, una técnica concreta, la técnica NÓESIS, y hemos analizado las particularidades, que desde el punto de vista de la metamodelización, conlleva el estudio de los lenguajes de modelización que han sido diseñados para representar comportamiento. En este capítulo vamos a poner en práctica la mayor parte de las ideas que se han mostrado en los capítulos precedentes, a través de la presentación de un metamodelo concreto de un formalismo de comportamiento

particular. Como hemos observado en el preámbulo de este capítulo, el formalismo de comportamiento que hemos elegido como ejemplo de trabajo es Statecharts. A la vista de la multitud de variantes del lenguaje que se han desarrollado en años recientes (véase [vdB94]), la primera decisión que era necesario tomar era cuál de dichas versiones iba a ser objeto de nuestro análisis. En este sentido, entendimos que lo más apropiado era considerar la versión del autor original del lenguaje, David Harel [Har87]. A pesar de ello, disponíamos de dos opciones, ya que Harel, junto con distintos miembros de su equipo de investigación, han presentado dos semánticas diferentes para Statecharts. La original se remonta al año 1987 y apareció publicada en el artículo [HPSS87]. Sin embargo, de manera paralela al desarrollo de la herramienta CASE STATEMATE [HLN+90], Harel presentó una nueva semántica para Statecharts, la cual se esboza en [HN96] y aparece con mayor detalle en [HP98]: es precisamente esta segunda versión la que ha sido recogida en el metamodelo que presentamos. A este respecto es necesario hacer notar que durante el proceso de desarrollo del metamodelo y paralelamente del presente trabajo, una nueva variante de Statecharts, bajo el nombre de State Machines, se ha incorporado como uno de los lenguajes agrupados dentro de UML. La creciente popularidad de UML como lenguaje de modelización en los últimos años nos ha invitado a usar UML State Machines como ejemplo, aunque sin renunciar a nuestra idea original. Por tanto en el presente capítulo mostramos un metamodelo NÓESIS del formalismo Statecharts de Harel, y en el siguiente capítulo mostraremos un metamodelo de UML State Machines. Las razones del interés del planteamiento diferenciado de estos dos ejemplos son diversas. En primer lugar, State Machines es una variante de Statecharts que no recoge toda la potencia del lenguaje original, ya que el contexto en el que se enmarca State Machines (el análisis y diseño orientados a objeto) hace que no sean necesarias algunas de las herramientas de las que dispone Statecharts. En segundo lugar, aunque la sintaxis y la semántica estática de State Machines han sido definidas de manera bastante precisa en [OMG01b], su semántica dinámica no está definida de una manera rigurosa, como varios autores han hecho notar [LP99a, LMM99, EHHS00], y por tanto el tipo de aportación que ha de realizar un metamodelo de UML State Machines es distinto del que ha de realizar un metamodelo de Statecharts. Por último, es necesario reconocer que una de las mayores ventajas de UML es el haber alcanzado una uniformización, un lenguaje compartido por múltiples tipos de usuario para el análisis y diseño orientado a objetos. Por tanto, para que la comunidad

UML acepte sin reservas un metamodelo de UML State Machines, éste ha de estar escrito siguiendo las reglas de escritura del metamodelo de UML, que en particular impone que cualquier metamodelo de UML debe estar expresado en UML. Como veremos en el siguiente capítulo, la flexibilidad de la arquitectura NÓESIS permite que el metamodelo de UML State Machines que proponemos esté expresado utilizando UML como lenguaje de metamodelización, si bien siguiendo las directrices marcadas por la arquitectura.

Así pues, el metamodelo NÓESIS de Statecharts que vamos a presentar en este capítulo recoge la versión del lenguaje presentada en [HN96, HP98]. Puesto que en efecto Statecharts es un formalismo para representar comportamiento, utilizaremos la definición de metamodelo de comportamiento NÓESIS que hemos presentado en el capítulo anterior. Así, la especificación del metamodelo contendrá todas las componentes que allí se citaban, y que resumidamente consisten en las siguientes: *perspectiva*, *sistema de referencia*, *marco representacional* y *definición de modelo*. Comenzaremos de inmediato presentando la perspectiva, ya que recordemos que lo que esta componente proporciona es una descripción básica del metamodelo por lo que es interesante presentarla inicialmente, centrando de esta forma el contexto. A continuación mostraremos el sistema de referencia, el cual proporciona una definición precisa de los conceptos necesarios para entender dicho metamodelo, y por tanto, y aunque habitualmente su construcción será progresiva, será una componente que también es conveniente presentar al principio para ser utilizada, como su propio nombre indica, como referencia. Las componentes de marco representacional y definición de modelo se presentarán a continuación, aunque será necesario hacer algunas precisiones acerca de su construcción.

4.2. Perspectiva y Sistema de Referencia

Así pues, como primer elemento del metamodelo, comenzamos presentando la *perspectiva* del metamodelo NÓESIS de Statecharts. Esta perspectiva, que se muestra en la tabla 4.1, se obtiene como resultado de una interpretación muy general del significado del lenguaje, a partir de las definiciones contenidas en [HN96, HP98].

Una vez determinada esta visión general y por tanto el contexto, en sentido amplio, en el que se presenta el metamodelo, en la tabla 4.2 se muestra el sistema de referencia del metamodelo NÓESIS de Statecharts. En él aparece la descripción

Tabla 4.1: Perspectiva del metamodelo NÓESIS de Statecharts

PERSPECTIVA DE STATECHARTS
Statecharts se corresponde con uno de los lenguajes del conjunto STATEMATE, utilizado para modelizar sistemas reactivos y basado en el paradigma estructural: vista funcional del sistema (descrita mediante activitycharts), vista de comportamiento (descrita mediante statecharts) y vista estructural (descrita mediante modulecharts). En particular los aspectos de comportamiento del sistema se especifican mediante statecharts, potencialmente uno por cada actividad en el activitychart. Los statecharts son una extensión de los clásicos diagramas estado–transición principalmente en tres sentidos: jerarquía, concurrencia y difusión de comunicación. La jerarquía permite que un estado sea descompuesto en varios subestados mediante AND u OR. Un estado que sigue una descomposición OR en varios subestados debe residir sólo en uno de sus subestados. Por el contrario, un estado que sigue una descomposición AND en varios subestados debe residir en todos sus subestados. Las transiciones entre estados pueden ser agrupadas. Una transición que se origina en el límite de un estado se aplica a todos sus subestados. Los conectores historia permiten que 'sea recordada' una visita previa a un estado. Los cambios que ocurren en un paso determinado tienen efecto en el siguiente paso, de tal forma que se cumple la propiedad de consistencia global. Los conectores de terminación permiten al statechart detener su actividad padre. Un statechart puede utilizar algunos mecanismos para controlar las actividades de la cual es responsable y para detectar su status.

rigurosa de cada concepto que se entiende esencial para la comprensión del metamodelo, y en particular se incluyen en él los conceptos necesarios para la correcta interpretación del marco representacional que mostraremos en una próxima sección.

Tabla 4.2: Sistema de referencia para Statecharts

sistema_reactivo *sistema que se caracteriza por estar, en gran medida, dirigido por eventos, teniendo que reaccionar continuamente a estímulos externos e internos*	[Har87, p.231]
actividad *componente funcional de un sistema reactivo*	[HP98, p.19]

continúa en página siguiente

viene de página anterior	
descripción_de_comportamiento *descripción de los aspectos dinámicos de un sistema reactivo completo o de una actividad particular, incluyendo control y timing*	[HP98, p.53]
DADO UN SISTEMA REACTIVO Y POSIBLEMENTE UNA ACTIVIDAD SUYA	
estado *estado completo o parcial del sistema_reactivo*	int.
acción *acción llevada a cabo por el sistema_reactivo que idealmente consume tiempo cero*	[Har87, p.256]
acción_asociada_con_entrada_a_estado *acción que tiene lugar al entrar a un estado*	[HN96, p.300]
acción_asociada_con_salida_de_estado *acción que tienen lugar al salir de un estado*	[HN96, p.300]
acción_asociada_a_estado *acción_asociada_ con_entrada_a_estado o acción_asociada_con_salida_de_estado*	
evento *señal de comunicación instantánea que indica que algo ha sucedido*	[HP98, p.43]
disparador *evento cuya ocurrencia provoca que reaccione el sistema_reactivo*	[HP98, p.54]
reacción *disparador y acción tales que la ocurrencia del disparador causa que la acción sea llevada a cabo*	[HP98, p.55]
reacción_estática_de_estado *reacción cuya acción será llevada a cabo (siempre que ocurra su disparador) mientras que el sistema_reactivo esté en (y no salga de) un estado*	[HN96, p.297]
actividad_activa_durante_estado *actividad que comienza a estar activa al entrar a un estado y deja de estarlo al salir de él*	[HN96, p.297]
actividad_activa_dentro_de_estado *actividad que comienza en algún momento durante el tiempo en que el sistema_reactivo está en un estado y se detiene cuando se sale del estado (a menos que la actividad se haya detenido antes)*	[HP98, p.109]
actividad_asociada_a_estado *actividad_activa_ durante_estado o actividad_activa_dentro_de_estado*	
continúa en página siguiente	

viene de página anterior	
elemento_de_comportamiento_de_estado *acción_asociada_a_estado o reacción_estática_de_estado o actividad_asociada_a_estado*	
comportamiento_asociado_a_estado *conjunto de elementos_de_comportamiento_de_estado*	
descomposición *descomposición de un estado en otros estados en modo AND u OR*	[HP98, p.59]
descomposición–or *descomposición de un estado *s* en otros estados de tal forma que estar en el estado *s* es estar exclusivamente en uno cualquiera de esos estados*	[HP98, p.59]
descomposición–and *descomposición de un estado *s* en otros estados de tal forma que estar en el estado *s* conlleva estar en todos esos estados simultáneamente*	[HP98, p.61]
estado_s1_es_padre_de_estado_s2 *estado *s1* ha sido descompuesto en un estado *s2* y posiblemente otros estados*	[HP98, p.59,61]
estado_s1_es_subestado_de_estado_s2 *estado_s2_es_padre_de_estado_s1*	[HP98, p.59,61]
estado_s1_es_descendiente_de_estado_s2 *estado_s1_es_subestado_de_estado_s2 o *s1* es un subestado de un estado que es un subestado de otro estado, y así hasta encontrar, después de un número finito de pasos, un estado que es subestado de *s2**	[HP98, p.60,23], int.
estado_s1_es_antecesor_de_estado_s2 *estado_s1_es_padre_de_estado_s2 o *s1* es padre de un estado que es padre de otro estado, y así hasta encontrar, después de un número finito de pasos, un estado que es padre de *s2**	[HP98, p.60,23], int.
estado–or *estado que está descompuesto en una descomposición–or*	[HP98, p.59]
estado–and *estado que está descompuesto en una descomposición–and*	[HP98, p.61]
componente_ortogonal_de_estado *estado que es subestado de un estado–and*	[HP98, p.61]

continúa en página siguiente

viene de página anterior	
raíz *estado que no tiene estado padre*	[HN96, p.297]
estado_básico *estado que no tiene subestados*	[HN96, p.297]
conector_terminación *estado virtual final de tal forma que la actividad deja de estar activa al entrar en él*	[HP98, p.104]
estado_s_es_padre_de_conector_terminación *conector_terminación que es tratado como estado_básico cuyo padre es el estado *s**	[HN96, p.316], int.
transición *posible cambio de estar en un conjunto de estados a estar en otro conjunto de estados y conectores_terminación que es disparado por la ocurrencia de un evento, a condición de que el evento exista, y que provoca que una acción sea llevada a cabo instantáneamente si y cuando el cambio suceda, a condición de que la acción exista*	[HP98, p.54,57,104]
conector_historia *entrada a un grupo de estados a través de la historia del sistema_reactivo en ese grupo*	[HP98, p.71]
estado–or_es_padre_de_conector_historia *conector_historia que es una entrada a los descendientes de un estado–or*	int.
conector_historia_de_estado–or *conector_historia que tiene un estado–or como padre*	[HN96, p.304], int.
conector_H *conector_historia que es entrada a los subestados de su padre s en los que estaba el sistema_reactivo la última vez que estaba en *s**	[HN96, p.304], [HP98, p.71], int.
configuración_relativa_a_estado_s *conjunto de estados C cumpliendo las siguientes reglas: C contiene al estado *s*; si C contiene un estado–or *s1*, debe contener exactamente un de los subestados de *s1*; si C contiene un estado–and *s2*, debe contener todo los subestados de *s2*; los únicos estados de C son los requeridos por las reglas anteriores*	[HN96, p.299]
configuración_básica_relativa_a_estado *conjunto de todos los estados_básicos de una configuración_relativa_a_estado*	[HN96, p.299]

continúa en página siguiente

viene de página anterior	
*conector_H** *conector_historia que es entrada a la configuración_básica_relativa_a su padre *s* en la que estaba el sistema_reactivo la última vez que estaba en s*	[HN96, p.304], [HP98, p.71], int.
conector_por_defecto *entrada por defecto a un estado–or*	[HP98, p.60], int.
estado–or_es_padre_de_conector_por_defecto *conector_por_defecto es entrada a estado–or*	int.
conector_hijo *conector_historia o conector_por_defecto o conector_terminación*	
condición *señal persistente que puede ser cierta o falsa*	[HP98, p.29]
conector–or *componente que ayudar a expresar brevemente varias transiciones, de tal forma que representa bifurcaciones mediante diferentes tipos de comportamiento*	[HP98, p.67], [HN96, p.302], int.
conector_condición *conector–or que representa bifurcación mediante condiciones*	[HP98, p.65,67]
conector_selección *conector–or que representa bifurcación mediante eventos*	[HP98, p.66,67]
conector_cruce *conector–or que no es ni un conector_condición ni un conector_selección*	[HP98, p.66]
conector_unión *componente que ayuda a representar salidas simultáneas de varias componentes_ortogonales_de_un_estado–and*	[HP98, p.69]
conector_bifurcación *componente que ayuda a representar entradas simultáneas a varias componentes_ortogonales_de_un_estado–and*	[HP98, p.69]
conector–and *conector_unión o conector_bifurcación*	[HN96, p.302]
conector *conector–or or conector–and or conector_hijo*	
estado|conector *estado o conector*	
disparador_de_transicion *disparador cuya ocurrencia es necesaria para que una transición se produzca*	[HP98, p.56]

continúa en página siguiente

viene de página anterior	
acción_de_transición *acción que se lleva a cabo simultáneamente cuando se produce una transición*	[HP98, p.57]
reacción_de_transición *reacción, disparador t y acción a, tal que t es el disparador de una transición r y a es acción de la transición r*	[HP98, p.57]
segmento_de_transición *conexión de un estado|conector a otro que está asociado opcionalmente a un disparador o una acción o a ambos, de tal forma que representa una transición t por sí misma o al combinarla con otros segmentos_de_transición de tal forma que el disparador_de_transición t, si existe, es la conjunción de los disparadores de los segmentos y la acción_de_transición t, si existe, es la concatenación de las acciones correspondientes*	[HN96, p.302]
origen_de_segmento_de_transición *estado|conector del que sale segmento_de_transición*	int.
destino_de_segmento_de_transición *estado|conector al que llega segmento_de_transición*	int.
disparador_de_segmento_de_transición **	int.
acción_de_segmento_de_transición **	int.

4.3. Conceptos básicos del metamodelo: estado y transición

Una vez establecidas las componentes de referencia es necesario hacer algunas precisiones acerca de los conceptos fundamentales del metamodelo, lo que nos permitirá conectar con la componente más destacada de todo metamodelo NÓESIS, el marco representacional. Puesto que Statecharts es una extensión de los diagramas estado-transición, los dos conceptos fundamentales que recoge son, obviamente, los conceptos de *estado* y *transición*. A pesar de ser los conceptos fundamentales, o quizá precisamente por ello, su definición precisa dista mucho de ser sencilla, y, aún peor, de ser clara.

En particular, resulta muy llamativo el observar que *en ninguna de las* dos *referencias* en las que nos hemos basado principalmente para elaborar este metamodelo,

es decir, [HN96] y [HP98], *existe una definición precisa del concepto de estado*. Ni siquiera en los artículos originales de definición del lenguaje, tales como [Har87], aparece una definición tal, sino que siempre se hace una referencia indirecta al hecho de que Statecharts está basado en los diagramas de estado-transición. Así, por ejemplo, podemos leer en [HP98, pág. 54]:

> "Una técnica natural para describir la dinámica de un sistema es usar una *máquina de estados finitos*. El sistema descrito o función está siempre en uno o en un conjunto finito de *estados*."

y, un poco más adelante, en [HP98, págs. 55-56], y bajo el epígrafe "Características básicas de Statecharts":

> "Como en los diagramas de estado-transición convencionales, los statechart se construyen básicamente a partir de estados y transiciones. Los estados en un statechart se dibujan como cajas rectilíneas con esquinas redondeadas."

De estos extractos y del resto de referencias citadas parece deducirse que el concepto de *estado* no ha sido definido de forma intencionada, probablemente por la dificultad de encontrar una definición sencilla de dicha noción. En efecto, siquiera un pequeño análisis del significado del término lleva con facilidad a terrenos más cercanos a la ontología o la filosofía que a aquél para el que se trata de utilizar el concepto, que es la modelización de comportamiento, y de sistemas reactivos en particular, pero siempre en el contexto de la informática. Por todo ello, y aunque en el sistema de referencia incluiremos una definición de estado (fundamentalmente por motivos de completitud) consideraremos el mismo planteamiento que hacen los autores de Statecharts, que es mantener la noción de estado como la idea intuitiva que cualquier lector pueda tener de ella.

Dejando pues a un lado las consideraciones más teóricas, entraremos a discutir aspectos más técnicos que afectan a la noción de estado desde el punto de vista de Statecharts. Una de las características más relevantes y novedosas del lenguaje Statecharts respecto a los tradicionales diagramas de estado-transición es la distinción entre estados-OR, estados-AND y estados básicos. Un *estado-OR* es un estado que se descompone en varios subestados, de tal forma que el que el sistema se encuentre

en el estado-OR significa que se encuentra en uno y sólo uno de sus subestados[1]. Un *estado-AND* es un estado que se descompone en varios subestados, de tal forma que el que el sistema se encuentre en el estado-AND significa que se encuentra en todos y cada uno de sus subestados. Un *estado básico* es un estado que no tiene subestados; dicho de otro modo, cualquier estado que no sea estado-OR o estado-AND es un estado básico. A la hora de (meta)modelar esta situación, la solución más sencilla (y que es similar a la que ha adoptado UML a la hora de definir State Machines, [OMG01b, pág. 2-147]) es considerar un concepto *estado* y tres especializaciones de dicho concepto, *estado-OR*, *estado-AND* y *estado básico*. Sin embargo entendemos que este no es el criterio más adecuado por dos razones, una de carácter teórico-ontológico y otra de carácter práctico, que sin embargo se encuentran bastante relacionadas. En primer lugar, y apelando a la idea intuitiva de la noción de estado, entendemos que el hecho de considerar que un estado se pueda descomponer o no en subestados no es una característica propia o que pertenezca a la esencia del estado. Un estado no es un estado diferente por el hecho de que se pueda o no descomponer en subestados. De hecho, durante el proceso de modelización, un diseñador puede modelar un estado que inicialmente es considerado como básico, pero que durante el proceso de creación del modelo se observe que es necesario que sea un estado-OR o un estado-AND, *sin que tal diseñador considere tal estado como un estado diferente*. En términos de bases de datos, esa transformación del estado no debería considerarse como una baja y un alta consecutivas, o una destrucción y una creación, sino simplemente una modificación. Este análisis fundamentalmente teórico se conjuga con la razón de carácter más práctico. En el caso de considerar un concepto estado y tres "subconceptos"de dicho concepto, el cambio de tipo de un estado provocaría, en un modelo construido a partir de dicho esquema, la eliminación de una instancia de un concepto y la creación de otra instancia de otro concepto: esta situación es habitual fuente de problemas, y así ha sido analizado y recogido, dentro de un contexto de modelización orientada a objeto, en [CZ97], donde esta situación recibe el nombre de *anomalía de reclasificación de objetos*. A la vista de este razonamiento, en el metamodelo NÓESIS de Statecharts hemos adoptado una

[1]Observemos que si adoptamos una postura rigurosa de los tradicionales operadores lógicos en los que se inspiran los nombres de los tipos de estados en Statecharts, los estados-OR -a la vista de su definición- deberían haber sido llamados con mayor propiedad estados-XOR. De hecho, en [Har87, HN96, HP98] se describen los estados-OR haciendo referencia al "clásico *O* exclusivo"

postura diferente para la (meta)modelización del concepto estado y sus diferentes tipos. Esta postura se muestra de manera explícita en el Soporte Independiente de Status (Figura 4.1), el cual es el soporte principal que se incluye en el marco representacional del metamodelo. En este soporte se observa, por una parte, el concepto *estado* sin especificación de ningún subconcepto del mismo, y por otra parte, la introducción del concepto *descomposición*, el cual es una aportación de nuestro metamodelo. El concepto descomposición es descrito a través de un *padre*, que es a su vez un estado, y un conjunto de *subestados*, que también son, obviamente, estados. Cada instancia del concepto descomposición proporciona, para un estado padre, el conjunto de sus subestados. Observemos que, desde el punto de vista estructural, no hay diferencia entre un estado-OR y un estado-AND (los dos contienen un conjunto de subestados), y que la diferencia entre ambos es fundamentalmente conceptual, o expresado con mayor precisión, es una diferencia que es dependiente del status, aun a pesar de que en la especificación gráfica de un statechart sea conveniente establecer una diferenciación visual entre ambos conceptos. Para diferenciar pues, en el soporte independiente de status, entre estados AND y OR, basta considerar dos subconceptos *descomposición-OR* y *descomposición-AND* del concepto descomposición. Esta forma de modelar resuelve satisfactoriamente los problemas antes expuestos sobre la reclasificación de los objetos. Si un estado inicialmente considerado como básico en una fase preliminar del proceso de modelización pasa a ser considerado un estado-OR (o análogamente un estado-AND), no hay que realizar ninguna reclasificación del estado, sino que basta con agregar una instancia adecuada del concepto descomposición. Hagamos notar también que la consideración del problema de la reclasificación de objetos tampoco nos ha llevado a posturas extremistas, sino que hemos intentado mantener una visión equilibrada. En efecto, con nuestra propuesta, si un estado pasara de ser considerado estado-OR a estado-AND (o viceversa), el problema de la reclasificación de objetos seguiría existiendo. Si hubiéramos intentado evitarlo a toda costa, hubiera bastado con utilizar el clásico truco utilizado en modelización conceptual para evitar o enmascarar la aparición de jerarquías ISA: describir con un atributo *tipo de descomposición* el concepto descomposición y no incluir los subconceptos descomposición-OR y descomposición-AND. Hay varias razones que nos han llevado a no optar por esta solución. La más importante de todas ellas es que el objetivo del metamodelo es el comunicar, de la manera más clara posible, los distintos conceptos incluidos en el lenguaje que se está especificando.

La inclusión del descriptor *tipo de descomposición* hubiera ocultado la existencia de los dos tipos de descomposición existentes, que no aparecerían de forma explícita en el soporte, lo que podría dar lugar a malas interpretaciones (por ejemplo relativas a cuántos tipos de descomposiciones habría, cuáles serían, etc.). Este hecho está íntimamente ligado con el de que los conceptos fundamentales de un soporte, y que son a partir de los cuales se crea un modelo, son los conceptos terminales. Así, los conceptos que se entienden como importantes son los de descomposición-OR y descomposición-AND, de manera que el concepto descomposición sirve casi exclusivamente como apoyo sintáctico para englobar los descriptores comunes (en este caso todos) de los conceptos principales. Veremos repetida esta situación a lo largo de la explicación de este metamodelo. Por último, y desde un punto de vista más práctico, entendemos que los casos de reclasificación entre estados-OR hacia estados-AND (o viceversa) pueden ser poco frecuentes. Tengamos en cuenta que a la vista del significado de uno y otro concepto, un cambio de ese estilo supondría una modificación muy importante en el statechart que se estuviera diseñando, pues como veremos el uso de estados-AND conlleva diversas consideraciones relacionadas con las transiciones implicadas. Sin embargo, la reclasificación de un estado básico hacia un estado-OR o estado-AND (que se evita con nuestra propuesta) es mucho más posible, si atendemos a un planteamiento de diseño top-down, en el que inicialmente se considerara un estado básico que en fases posteriores del diseño pudiera ser refinado a ser algo más complejo. De hecho este tipo de razonamientos es el que motivó originalmente la inclusión de los estados-OR en Statecharts (véase, por ejemplo, [Har87, pág. 235]). Para concluir con las observaciones referentes al concepto estado, observemos que en el soporte el concepto en sí aparece descrito por un *nombre*, y opcionalmente un *comportamiento asociado*. Este último concepto, no trivial, necesita de la descripción correspondiente del sistema de referencia para comprenderse plenamente.

Una vez analizado el tratamiento del concepto estado dentro del metamodelo Nóesis de Statecharts, vamos a considerar ahora el otro concepto básico del lenguaje, el concepto de *transición*. De nuevo nos encontramos con una situación análoga a la anteriormente descrita para el concepto de estado, ya que en ninguna de las referencias podemos encontrar una definición más o menos precisa de lo que es una transición. Sin embargo, a partir de diferentes fragmentos es posible construir una descripción relativamente clara de lo que es una transición, al menos en

su acepción más elemental. Esta descripción aparece en el sistema de referencia del metamodelo (tabla 4.2). Es necesario hacer notar que en el soporte independiente de status no aparece el concepto transición, sino el concepto *segmento de transición*, del cual también se puede ver su descripción en el sistema de referencia. Esta aparente contradicción es debida a la complejidad del concepto transición. Una transición supone, de manera básica, el cambio de estar en un conjunto de estados a estar en otro. Sin embargo, por una parte el que esos estados implicados puedan ser estados-OR y/o estados-AND, y por otra el que aparezcan diversos tipos de conectores, hace que una transición sea en realidad algo bastante complejo, que conlleva la conexión de (en general) varios segmentos de transición, entendidos estos como los enlaces entre estados y conectores. En algunos casos un sólo segmento de transición será en efecto una transición, pero en otros casos los segmentos de transición se combinan a través de distintos tipos de conectores (los analizaremos un poco más adelante) para obtener lo que se denomina en [HN96, pág. 302] una *transición completa*, y que coincidiría con el concepto que nosotros hemos denominado simplemente transición. La razón fundamental por la que en el soporte independiente de status sólo se ha considerado el concepto segmento de transición y no así el de transición es doble. Por una parte, recordemos que en el soporte independiente de status se recogen todas aquellas nociones que no dependen de cuál sea la situación particular en que se encuentre un modelo statechart en un momento dado. De alguna manera, por tanto, la información recogida en este soporte se encuentra relacionada con *aquello que se expresa explícitamente al escribir el modelo*. En el caso del diseño de un statechart, a la hora de dibujar (recordemos que Statecharts es un formalismo visual) una transición, en realidad se dibujan los segmentos de transición, es decir, las conexiones entre estados y/o conectores, de manera que estos son los elementos realmente independientes del status. La determinación de las transiciones completas, es decir, las construidas como combinación de uno o más segmentos de transición, es una tarea que debe realizarse al analizar la información dependiente del status. Esto es así porque pueden seguirse diferentes estrategias para la determinación de las transiciones completas, en relación con cuáles de ellas sean o no disparables, lo cual, obviamente, afecta al status concreto en que se encuentre el sistema, el tipo de conectores implicados en cada transición completa concreta, etc. Trataremos de nuevo el concepto de transición más adelante. Por el momento nos concentraremos en la descripción del concepto más básico, es decir, segmento de transición.

Previamente a esta descripción es necesario hacer algunos comentarios sobre el concepto *conector*. Los conectores son componentes que se utilizan en Statecharts básicamente con dos objetivos: o bien se utilizan para facilitar la escritura de múltiples transiciones o bien para representar ciertos tipos de entradas a estados y terminación de ejecuciones. Fundamentalmente debido al primero de estos dos casos, los segmentos de transición se trazan precisamente entre estados y conectores (en la mayor parte de los casos los segmentos se trazarán entre un estado y otro o entre un estado y un conector -o viceversa-, pero en algunos casos también pueden aparecer segmentos entre conectores). Esta es la razón por la que en el metamodelo se incluye un concepto auxiliar *estado|conector*. Observemos como en el Soporte Independiente de Status se incluye este concepto, a través de un pequeño cuadrado en blanco con el nombre del concepto junto a (y fuera de) dicho cuadro. Esta representación gráfica pretende significar el hecho de que este concepto no es proporcionado por el lenguaje modelado, sino que se crea en el metamodelo con la única intención de facilitar la descripción de otros conceptos o restricciones del metamodelo. A partir de esta observación, el concepto segmento de transición se describe por medio de tres conceptos atributivos: origen, destino y reacción. Como ya hemos comentado, el origen y el destino de un segmento de transición son estados o conectores, y representan el elemento desde el que, y hasta el que, respectivamente, se traza el segmento de transición. Con respecto al concepto reacción, la descripción del concepto que aparece en el Sistema de Referencia es aparentemente bastante precisa, aunque necesita el apoyo de la información recogida en el Soporte de Independencia de Status para comprenderse plenamente. Observemos que en dicho soporte el concepto reacción se describe a través de una atribución-OR, tras la que aparecen los conceptos de *disparador* y *acción*. Tal y como se ha explicado en la introducción a la técnica NÓESIS en el Capítulo 2 el significado de una atribución-OR se corresponde de manera natural con el significado del operador OR de la lógica: la atribución tendrá como atributo al menos uno de los conceptos atributivos que le siguen. En el caso concreto de la atribución-OR del concepto reacción, una instancia de dicho concepto estará descrita por un disparador o por una acción. Puesto que se trata de un OR (no exclusivo), también se recoge el caso en el que una reacción venga descrita por un disparador y una acción.

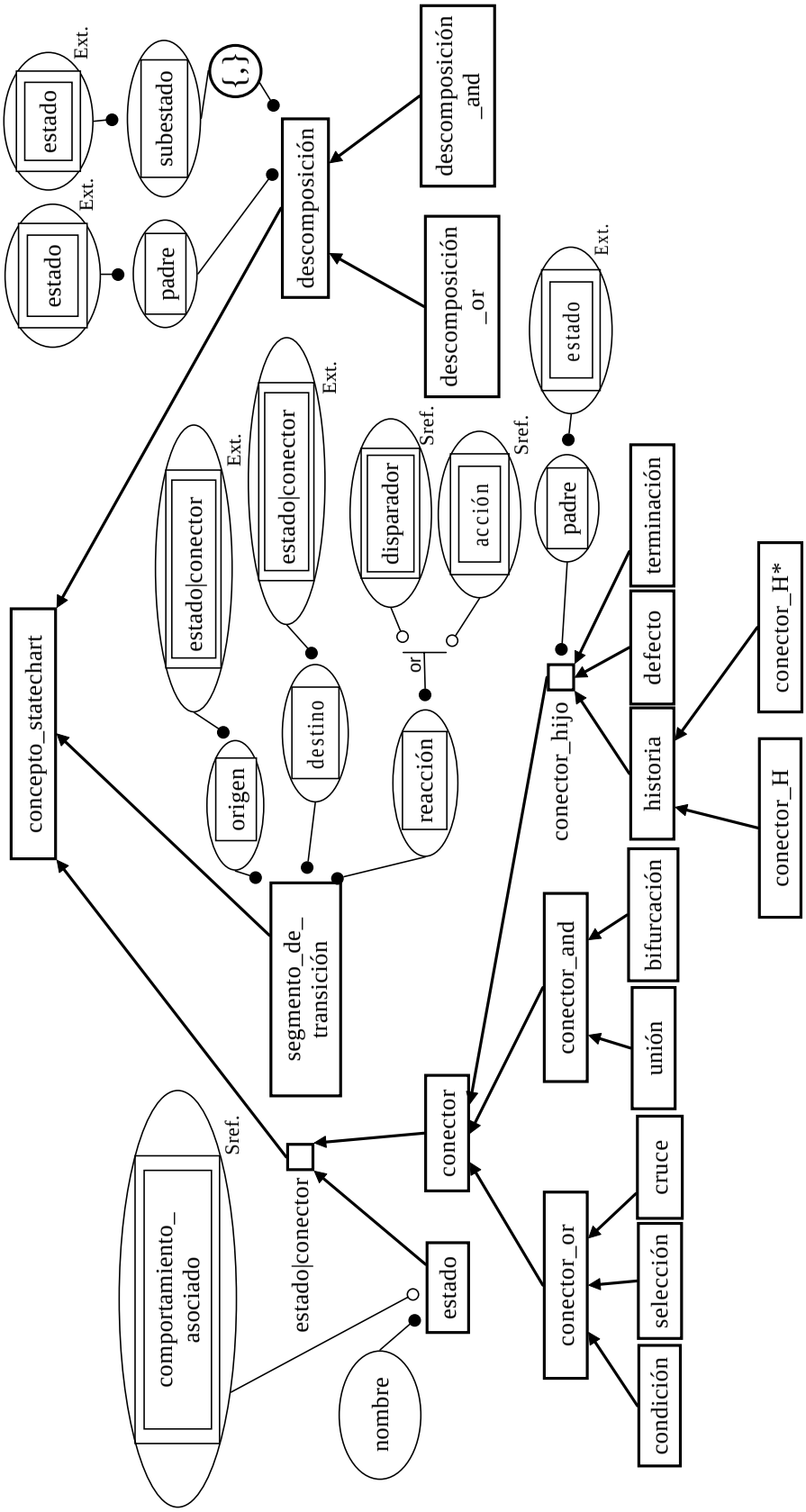


Figura 4.1: Soporte Independiente de Status del Metamodelo NÓESIS de Statecharts

4.4. Marco representacional

Tal y como hemos mostrado en las descripciones de la sección anterior, para mostrar plenamente el significado de los conceptos esenciales del metamodelo hemos necesitado apoyarnos en el sistema de referencia pero también en el soporte que forma parte del marco representacional. Según la definición genérica de metamodelo de comportamiento NÓESIS, el marco representacional se alinea con el nivel independiente del status de la arquitectura NÓESIS. Por ello, en el metamodelo NÓESIS de Statecharts el soporte incluido en el marco representacional se denomina soporte independiente de status, y aparece en la figura 4.1. Dicho marco representacional se completa con un conjunto de restricciones locales, que mostramos en la tabla 4.3.

Tabla 4.3: Restricciones locales para Statecharts

[L1] No pueden existir dos descomposiciones diferentes de un mismo estado.	
[L2] Un estado es subestado de como máximo un estado.	
[L3] Un estado no puede ser descendiente de sí mismo.	
[L4] Un estado es padre de como máximo un conector_H.	
[L5] Un estado es padre de como máximo un conector_H*.	
[L6] Un estado es padre de como máximo un conector_por_defecto.	
[L7] Un estado es padre de como máximo un conector_terminación.	
[L8] Un conector_terminación no tiene salidas.	[HP98, p.104]
[L9] Un conector_por_defecto no puede ser destino_de_segmento_de_transición.	
[L10] Un conector_condición es el origen de dos o más segmentos_de_transición de tal forma que cada uno tiene asociado un disparador que sucede cuando una condición es verdadera.	[HP98, p.58]
[L11] Un conector_unión es origen de exactamente un segmento_de_transición.	

continúa en página siguiente

viene de página anterior
[L12] Un conector_unión es destino de dos o más segmentos_de_transición cuyos orígenes son cada uno de ellos descendientes de una componente ortogonal de un mismo estado–and. *[L13]* Un conector_bifurcación es destino de exactamente un segmento_de_transición. *[L14]* Un conector_bifurcación es origen de dos o más segmentos_de_transición cuyos destinos son cada uno de ellos descendientes de una componente ortogonal de un mismo estado–and.

4.5. Definición de modelo

Observemos que hasta llegar a este punto el metamodelo que estamos presentando no difiere esencialmente de la definición básica de metamodelo NÓESIS que hemos presentado en el Capítulo 2, y por tanto, desde un punto de vista meramente estructural, es análogo al ejemplo que hemos mostrado en dicho Capítulo mediante el metamodelo de RM/T. A la vista de la noción de metamodelo de comportamiento NÓESIS que hemos descrito en el Capítulo 3, las diferencias fundamentales en el metamodelo aparecen en la componente de definición de modelo. Recordemos que en esta componente era necesario especificar en primera instancia un conjunto de soportes de status, una transformación T_0 y una transformación T, teniendo cada uno de estos elementos el significado que se ha explicado en el Capítulo 3.

En concreto, en este caso hemos limitado el conjunto de soportes de status a un único soporte, denominado Soporte de Status del metamodelo NÓESIS de Statecharts (figura 4.2). Comparando este Soporte con el Soporte de la figura 4.1 es claro que el Soporte de Status es más complejo que el Soporte Independiente de Status. Básicamente esto sucede porque un 'modelo' (entendido como conjunto de instancias de los conceptos fuente junto con las restricciones locales aplicables) del Soporte Independiente de Status se corresponde con la noción habitual de statechart, en el sentido de un diagrama (aunque recordemos que los metamodelos NÓESIS no recogen aspectos puramente representacionales) de estados y transiciones que representa un comportamiento genérico. Por el contrario, un 'modelo' del Soporte de Status representa un statechart en una situación concreta, en un momento concreto de su ejecución, de su comportamiento, y por tanto son necesarios más conceptos

y descriptores para especificarlo. Destaquemos en particular que en el Soporte de Status se ha incluido la información referente a los *datos de entrada*, es decir, la información referida a los eventos, condiciones y valores primitivos. Además de esto el otro concepto fundamental que aparece en el Soporte de Status es el de *transición compuesta completa*. Este es el concepto que verdaderamente se corresponde con la idea más general de transición. Recordemos que en el Soporte Independiente de Status se había incluido exclusivamente un concepto de *segmento de transición* entendido como aquel trazado entre estados y/o conectores de forma elemental. Sin embargo, la construcción de transiciones completas se realiza sólo durante el análisis del comportamiento particular del statechart, durante la ejecución del statechart. La definición correcta de una transición compuesta completa integra la consideración combinada de las descomposiciones de estados, así como los distintos tipos de conectores, en particular los conectores AND y OR o los conectores de Historia. Precisamente los conectores de Historia han de ser descritos, en el Soporte de Status, con los conceptos referentes a la configuración básica de estados de la situación anterior.

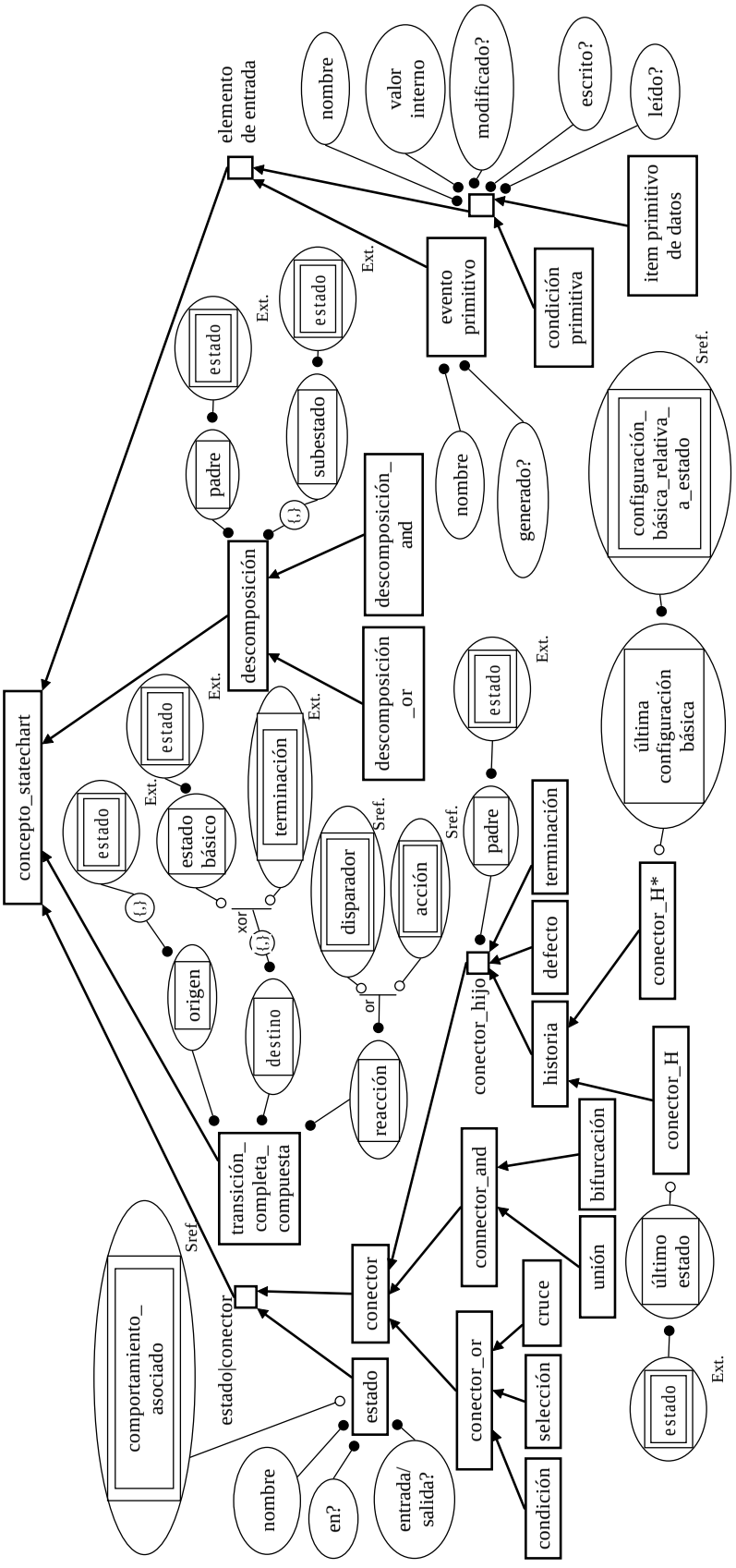


Figura 4.2: Soporte de Status del Metamodelo NÓESIS de Statecharts

Por su parte, las transformaciones T_0 y T se describen en modo semi–textual en la tabla 4.4, con el nombre concreto, ya que pertenecen a este metamodelo concreto, de $T_0 - statecharts$ y $T - statecharts$. La transformación $T_0 - statecharts$ determina la forma de pasar de un modelo del Soporte Independiente de Status a un modelo del Soporte de Status y representa el paso de un statechart "estático" a un statechart "en ejecución", en concreto en su situación inicial. Por ello en la especificación de la transformación el primer argumento es un modelo de tipo 'statechart estático', y el resultado de la transformación es un 'statechart dinámico'. La transformación $T - statecharts$ por su parte, determina la forma de pasar de un modelo del Soporte de Status a otro modelo del Soporte de Status, representando el cambio de una situación actual del statechart a la siguiente (de ahí que en la especificación de la transformación el primer argumento sea un 'statechart dinámico' y su resultado sea otro 'statechart dinámico'). Esta transformación se corresponde de hecho con la descripción algorítmica de la noción de *paso* que se presenta tanto en [HN96] como en [HP98]. Como ya ha sido observado anteriormente, la tarea representada en la transformación T en general, para cualquier formalismo de comportamiento, es extremadamente compleja. Así ocurre para el formalismo Statecharts, y se hace imprescindible utilizar la noción de refinamiento de la arquitectura NÓESIS para poder especificar con más detalle la transformación T. En concreto, nos hemos inspirado en la estructura del algoritmo al que hemos hecho referencia, introduciendo un refinamiento de la transformación T en tres fases. Estas tres fases se traducen en la introducción de tres Soportes de Status Virtual, que mostramos en la figura 4.3 (Soporte de Status Virtual 1), la figura 4.4 (Soporte de Status Virtual 2) y la figura 4.5 (Soporte de Status Virtual 3). La inclusión de estas fases intermedias conlleva la correspondiente introducción de cuatro transformaciones de modelos, $T_i - statecharts$, $i = 1, 2, 3, 4$, que mostramos también en la tabla 4.4. Estas transformaciones incluyen argumentos y resultados del tipo 'statechart de status virtual i', los cuales son modelos de los Soportes de Status Virtual respectivos. Observemos que estas transformaciones verifican la propiedad exigida en la noción básica de refinamiento de la arquitectura NÓESIS, propiedad que obliga a que la transformación $T_4 \circ T_3 \circ T_2 \circ T_1$ dé como resultado la transformación T. En particular, el primer argumento de la transformación T_1 y el resultado de la transformación T_4 son modelos del Soporte de Status, es decir, son 'statecharts dinámicos'.

Tabla 4.4: Transformaciones para Statecharts

$T_0 - statecharts(M$: statechart estático; P: conjunto de elementos de entrada)	
Obtiene:	M_D^0, statechart dinámico
Descripción:	Para cada instancia de *estado* in M_D^0, el valor del atributo *en?* se calcula a partir de las instancias de *descomposición* y *conector_por_defecto* en M. Para cada instancia de *estado* en M_D^0, el valor del atributo *entrada/salida?* se establece a 'falso'. Las instancias de *transición_compuesta_completa* en M_D^0 se calculan a partir de las instancias de *segmento_de_transición* y otras instancias en M. Las instancias de *conector_historia* en M_D^0 se establecen a 'vacío'. Las instancias de *elemento_de_entrada* en M_D^0 se calculan a partir de P.
$T - statecharts(M_D$: statechart dinámico; P: conjunto de elementos de entrada *externos*)	
Obtiene:	$M_D^{'}$, statechart dinámico
Descripción:	Ejecuta el algoritmo básico de paso y calcula los nuevos descriptores de las instancias de *estado*, *conector_historia* y *elemento_de_entrada* en $M_D^{'}$ a partir de los respectivos en M_D y P.
$T_1 - statecharts(M_D$: statechart dinámico; P: conjunto de elementos de entrada *externos*)	
Obtiene:	M_1, statechart de *status virtual 1*
Descripción:	Las instancias de *elemento_de_status* en M_1 se calculan a partir de las instancias de *elemento_de_entrada* en M_D y P.
$T_2 - statecharts(M_1$: statechart de *status virtual 1*)	
Obtiene:	M_2, statechart de *status virtual 2*
Descripción:	Las instancias de *paso* en M_2 se calculan a partir de las instancias de *transición_compuesta_completa*, *comportamiento_asociado* y *elemento_de_status* en M_1.

continúa en página siguiente

viene de página anterior	
$T_3 - statecharts(M_2$: statechart de *status virtual 2*)	
Obtiene:	M_3, statechart de *status virtual 3*
Descripción:	Los nuevos descriptores de las instancias de *estado*, *conector_historia* y *elemento_de_status* en M_3 se calculan a partir de sus instancias respectivas en M_2 y a partir de las instancias de *paso* en M_2.
$T_4 - statecharts(M_3$: statechart de *status virtual 3*)	
Obtiene:	$M_D^{'}$, statechart dinámico
Descripción:	Los nuevos descriptores de las instancias de *estado* y *conector_historia* en $M_D^{'}$ se calculan a partir de sus instancias respectivas en M_3 y las instancias de *elemento_de_entrada* en $M_D^{'}$ se calculan a partir de las instancias de *elemento_de_status* en M_3.

Respecto a los Soportes de Status Virtual es necesario hacer algunos comentarios. En primer lugar hay que destacar que en uno de ellos, en concreto en el Soporte de Status Virtual 2 (figura 4.4) aparece de forma explícita el concepto de *paso*. Tal y como hemos comentado este es un concepto clave para la descripción del comportamiento de los Statecharts. Nuestro enfoque tiene la ventaja de que conceptos claves como éste pueden aparecer de forma explícita, con lo que es posible describirlo y analizarlo detalladamente. Por otra parte, en los tres Soportes de Status Virtual se ha utilizado un aspecto de la notación de los soportes NÓESIS que no había sido necesaria hasta ahora. En el Capítulo 2 hemos mostrado en la tabla 2.2 (página 57) la notación en forma de rectángulo con línea discontinua, representando conceptos de soporte que aparecen en un soporte previo. Esta notación se ha utilizado en los Soportes de Status para simplificar la lectura de estos soportes. Puesto que los mismos deben incluir la práctica totalidad de los conceptos que aparecen en el soporte inmediatamente anterior, el uso de la notación de conceptos encerrados en línea discontinua permite identificar de forma sencilla aquellos conceptos cuya descripción es exactamente idéntica (siempre con respecto al soporte inmediatamente anterior), propiedad que se hereda para todos los subconceptos del concepto en cuestión. Si la descripción del concepto se modifica, sea del modo que sea, entonces el concepto vuelve aparecer de forma explícita en el soporte correspondiente. Por último, des-

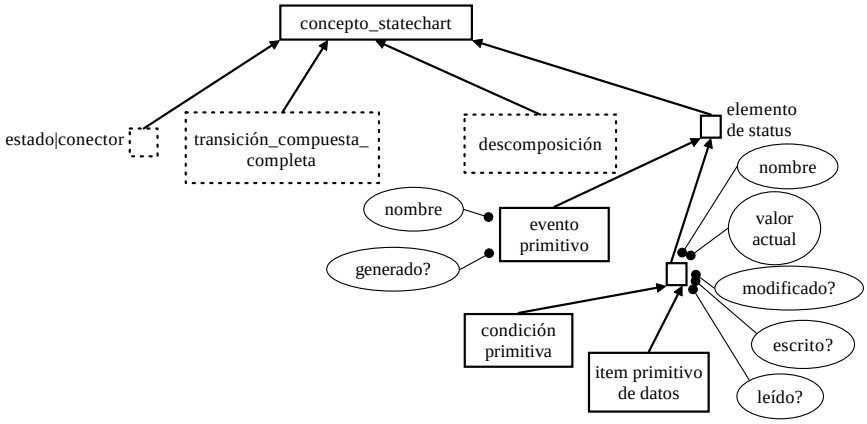


Figura 4.3: Soporte de Status Virtual 1 del Metamodelo NÓESIS de Statecharts

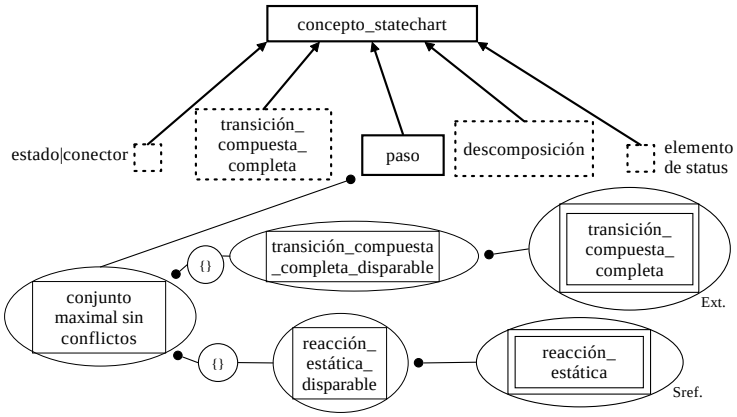


Figura 4.4: Soporte de Status Virtual 2 del Metamodelo NÓESIS de Statecharts

taquemos una de las ventajas del uso de este tipo de refinamientos. En concreto, el refinamiento de la transformación $T - statecharts$ muestra de forma clara la diferencia principal entre las versiones de Statecharts en [HPSS87] y en [HN96, HP98], que es «si los cambios que ocurren en un paso dado [...] deben tener efecto en el paso en curso o en el siguiente» [HN96]. En la versión de [HN96, HP98], que es la que recogemos en nuestro metamodelo, se asume que los cambios tienen efecto en el siguiente paso, circunstancia que es fácilmente detectable en nuestro metamodelo. En concreto, los cambios que ocurren en un paso se muestran en el Soporte de Status Virtual 3, y se utilizan en la preparación del siguiente paso (lo que se recoge en la transformación $T_4 - statecharts$), lo que sucede después de que la ejecución del paso en curso haya sido completada (transformación $T_3 - statecharts$).

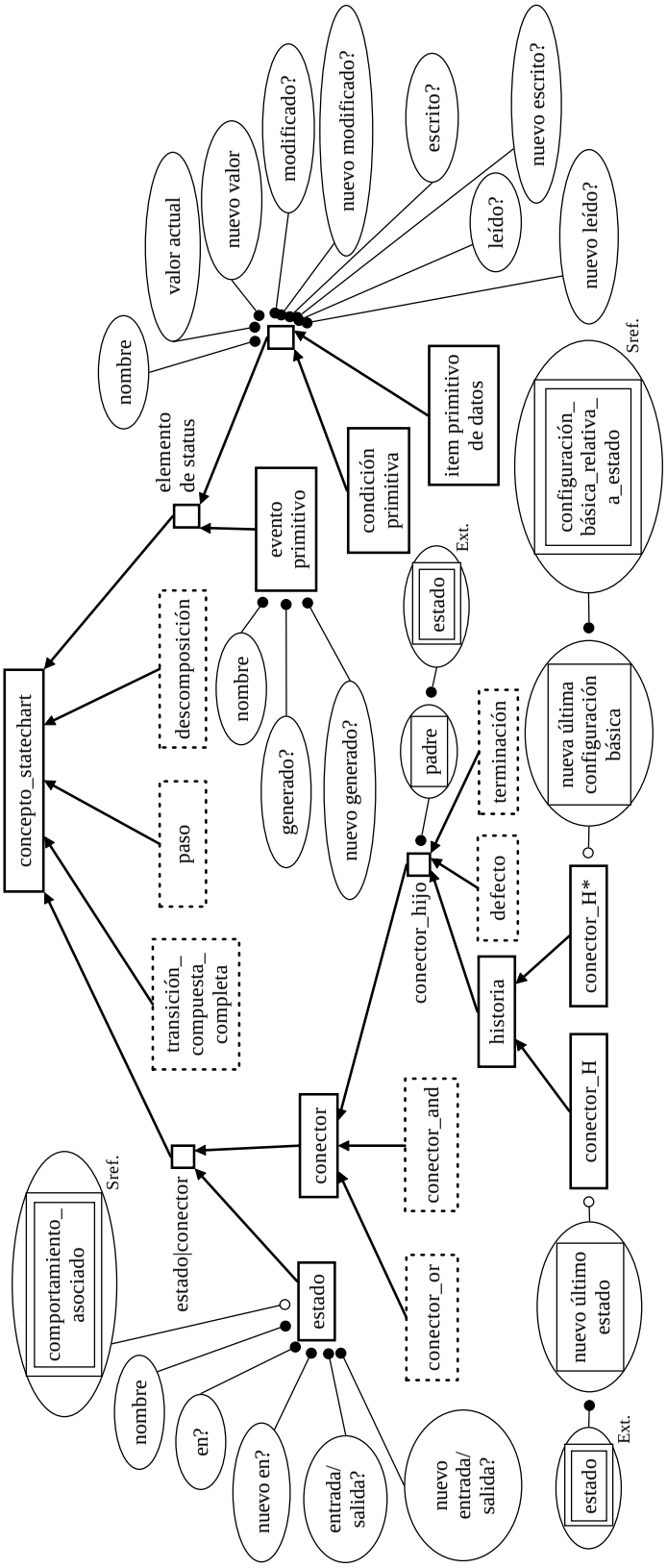


Figura 4.5: Soporte de Status Virtual 3 del Metamodelo NÓESIS de Statecharts

Capítulo 5

Un metamodelo UML de UML State Machines

En el capítulo anterior hemos mostrado un metamodelo del formalismo para la representación del comportamiento de sistemas reactivos conocido como Statecharts. Este metamodelo se ha expresado utilizando la técnica de metamodelización NÓESIS, técnica que hemos introducido también en el presente trabajo mediante la presentación de un ejemplo detallado, un metamodelo del modelo de bases de datos RM/T. Para la realización del metamodelo NÓESIS de Statecharts hemos hecho uso de una noción ampliada de la noción básica de metamodelo NÓESIS. Esta noción ampliada ha sido necesaria para ser capaces de especificar de forma fiel aquellos elementos que caracterizan al formalismo Statecharts como formalismo para la representación de aspectos de comportamiento. En particular, la noción ampliada de metamodelo NÓESIS se basa por una parte en la noción básica que hemos presentado en el Capítulo 2, y por otra en la arquitectura NÓESIS para la representación de comportamiento que hemos presentado en el Capítulo 3. El uso que hemos hecho hasta este momento de las distintas ideas que hemos venido presentando puede llevar a la conclusión de que estos conceptos son sólo aplicables al caso concreto de la técnica de metamodelización NÓESIS. Sin embargo, la intención y aplicabilidad es precisamente la contraria. La arquitectura NÓESIS establece un marco conceptual, genérico, no ligado específicamente ni con ningún formalismo o lenguaje para la representación de comportamiento concreto, ni, lo que es más, con *ninguna técnica de metamodelización particular*.

Prueba de ello es el metamodelo que mostramos en este capítulo. Este es un

metamodelo de la variante de Statecharts que bajo el nombre *State Machines* ha sido incluida como sub-lenguaje dentro de UML [BRJ99]. Este metamodelo usa la arquitectura NÓESIS como marco para su desarrollo, pero no está escrito usando la técnica NÓESIS, sino que utiliza el propio lenguaje UML como técnica de metamodelización. La razón para esta elección es que la especificación del propio lenguaje UML, es decir, el metamodelo de UML [OMG01b], *está escrito utilizando UML como lenguaje de metamodelización*. Esta noción de *definición circular* ya ha sido analizada en los Preliminares del presente trabajo, y es comentada en el documento de especificación de UML diciendo que «[los autores] reconocemos que existen límites teóricos acerca de lo que uno puede expresar sobre un metamodelo usando el propio metamodelo. Sin embargo, nuestra experiencia sugiere que esta combinación alcanza un equilibrio razonable entre expresividad y legibilidad» [OMG01b, p. 2-4]. A la vista de esta postura "oficial" en este capítulo utilizamos UML como lenguaje de metamodelización, lo que no supone un problema adicional debido a la flexibilidad de aplicación de la arquitectura NÓESIS.

5.1. Introducción al metamodelo

La técnica statechart es un formalismo visual definido como mejora de las máquinas de estados finitos, desarrollada originalmente por D. Harel [Har87] para la especificación de sistemas reactivos complejos. Mucho se ha escrito sobre este tópico en los últimos años, y en particular, se han propuesto un gran número de variantes de la técnica [vdB94]. Más recientemente, el éxito del formalismo statechart ha recibido un impulso adicional ya que una adaptación orientada a objeto de la técnica con el nombre *State Machines*, ha sido adoptada como parte del Lenguaje Unificado de Modelización (UML) [OMG01b, RJB99].

Un defecto conocido de UML State Machines es que en el documento de especificación de UML [OMG01b], si bien la sintaxis y la semántica estática de State Machines están expuestas de forma precisa, la semántica dinámica no está definida rigurosamente [LMM99, LP99a, EHHS00]. Sin duda, una especificación precisa del comportamiento de las State Machines es esencial para un amplio y variado grupo de personas. Por ejemplo, los usuarios finales del lenguaje (tales como analistas y diseñadores de sistemas) necesitan de al menos una idea general, aunque precisa, de cómo se comporta un state machine. En segundo lugar, los construc-

tores y desarrolladores de herramientas CASE que estén interesados en que sus aplicaciones permitan utilizar State Machines se beneficiarían de manera clara de una especificación no ambigua del lenguaje. Por último, los ingenieros del método usarían una especificación precisa de State Machines para analizar problemas tales como adaptabilidad del lenguaje, comparación con otros lenguajes para la representación del comportamiento, transformaciones, etc. Esta compleja situación ha provocado que la definición de una semántica dinámica precisa para State Machines sea un tema de intensa investigación en fechas recientes (véanse por ejemplo [MK98, LMM99, LP99b, RKR+00, BCR00, EHHS00]).

El problema es que la mayoría de los enfoques que tratan de determinar una semántica dinámica precisa para State Machines utilizan notaciones formales, notaciones fundamentalmente matemáticas, como por ejemplo [LMM99, LP99b, BCR00]. Este tipo de enfoque suele ser difícil de leer y de entender para la mayoría de los potenciales usuarios. Por esta razón, del mismo modo que otros autores [EHHS00], creemos que la presentación de la semántica dinámica mediante el uso de notaciones formales no es la solución más adecuada, ya que es indudable que la semántica dinámica debe ser establecida de un modo preciso, pero esta tarea debe realizarse de tal modo que se facilite la comprensibilidad y legibilidad de la especificación.

Hay que precisar que con esto no estamos negando la conveniencia del uso de notaciones formales para la resolución de cierto tipo de problemas importantes, tales como verificación o chequeo de modelos. Sin embargo en este trabajo proponemos adoptar una perspectiva de metamodelización para la especificación de la semántica dinámica, ya que se ha mostrado en la literatura [Ver93, tHV97] que esta perspectiva es un medio adecuado para la mejora de las propiedades de comprensibilidad y legibilidad. En concreto nuestra propuesta se basa en la arquitectura que hemos presentado en el Capítulo 3, arquitectura que establece de forma explícita una distinción entre el comportamiento genérico representado en un modelo dinámico (Nivel de Independencia de Status) y el comportamiento representado en relación con una situación particular (Nivel de Status). De forma adicional, el concepto de 'movimiento de la situación actual a la siguiente' es recogida utilizando la noción de transformación. Tomando esta arquitectura como base, el metamodelo de UML State Machines que proponemos consiste básicamente en dos diagramas de clases UML (un diagrama por cada uno de los niveles de la arquitectura) y dos transformaciones. Estas transformaciones representan, respectivamente, la determinación del

status inicial y el proceso llevado a cabo por un paso *run–to–completion* tal y como es definido en la semántica de UML. Ni en la definición actual de UML ni en la de MOF (Meta Object Facility [OMG01a], considerado como el meta–metamodelo de UML [OMG01b, p. 2–5]), se incluye ninguna noción de transformación entre diagramas de clases. Sin embargo, nociones similares han sido presentadas por diversos autores [Ver93, MHR96, tHV97, DZ00, DZE00] como artefactos necesarios para resolver problemas parecidos dentro del campo de la ingeniería del método, problemas tales como la interoperabilidad de métodos o la adaptación de métodos.

Recordemos, tal y como se ha mostrado en capítulos anteriores, que el uso de una perspectiva de metamodelización nos sitúa en un nivel que es independiente del caso concreto de State Machines, y por tanto nuestro enfoque es aplicable a otras técnicas de representación de aspectos de comportamiento. Sin ir más lejos, de los nueve tipos de diagramas que forman parte de UML, cinco de ellos están involucrados en la representación de algún tipo de comportamiento, si bien es cierto que State Machines en particular es el tercer tipo de diagrama más complejo dentro de UML (el más complejo, con amplia diferencia, son los Diagramas de Clases, seguidos por los Diagramas de Componentes, según se muestra en el estudio de complejidad elaborado por Siau y Cao en [SC01]). Pero sin duda UML no es el único lenguaje que trata aspectos de representación de comportamiento. A modo de ejemplo citaremos el estudio de Palanque y otros en [PBDSB93], en el cual, y limitándose al ámbito de la modelización de sistemas reactivos, se distinguen tres enfoques para la representación de este tipo de sistemas. Estos enfoques de modelización son denominados *basados en estados*, *basados en eventos* y *basados en Redes de Petri*, y existen distintas técnicas (o variantes de otras técnicas) que se han desarrollado siguiendo uno u otro paradigma. De esta diversidad nace el interés de disponer de una infraestructura que permita representar los aspectos de comportamiento de tal forma que esta infraestructura sea independiente del lenguaje, técnica, método o formalismo que se esté considerando.

5.2. Un metamodelo UML de UML State Machines

En las sub-secciones siguientes presentamos un metamodelo de UML State Machines, expresado en UML, que recoge la sintaxis, la semántica estática y la semántica dinámica del sub-lenguaje, tomando como base la arquitectura NÓESIS.

5.2.1. Nivel de Independencia de Status

Recordemos que dentro de la arquitectura NÓESIS el Nivel de Independencia de Status está relacionado con aquellos aspectos del comportamiento que son independientes de las distintas situaciones concretas, y que por tanto captura los aspectos genéricos de la representación de comportamiento. Por ello, para el caso particular del metamodelo de UML State Machines, el Nivel de Independencia de Status se corresponde con la sintaxis y la semántica estática del lenguaje. Para representar estas características, y a semejanza de lo que se propone en el Documento de Especificación de UML, en nuestro metamodelo proponemos utilizar un diagrama de clases UML que hemos llamado *Diagrama Estático* (figura 5.1) y un conjunto de expresiones escritas en OCL, Object Constraint Language (Lenguaje de Restricción de Objetos) [WK99], el lenguaje utilizado en UML para la expresión semi–formal de restricciones. Para realizar el Diagrama Estático hemos utilizado como punto de partida el diagrama de clases de UML State Machines que se propone en el Documento de Especificación de UML [OMG01b, p. 2–147], modificándolo en diferentes aspectos. Estas modificaciones permiten que el Diagrama Estático recoja explícitamente ciertas restricciones que un state machine debe cumplir, para lo cual hemos tenido en cuenta también el metamodelo NÓESIS de Statecharts que hemos presentado en el capítulo anterior.

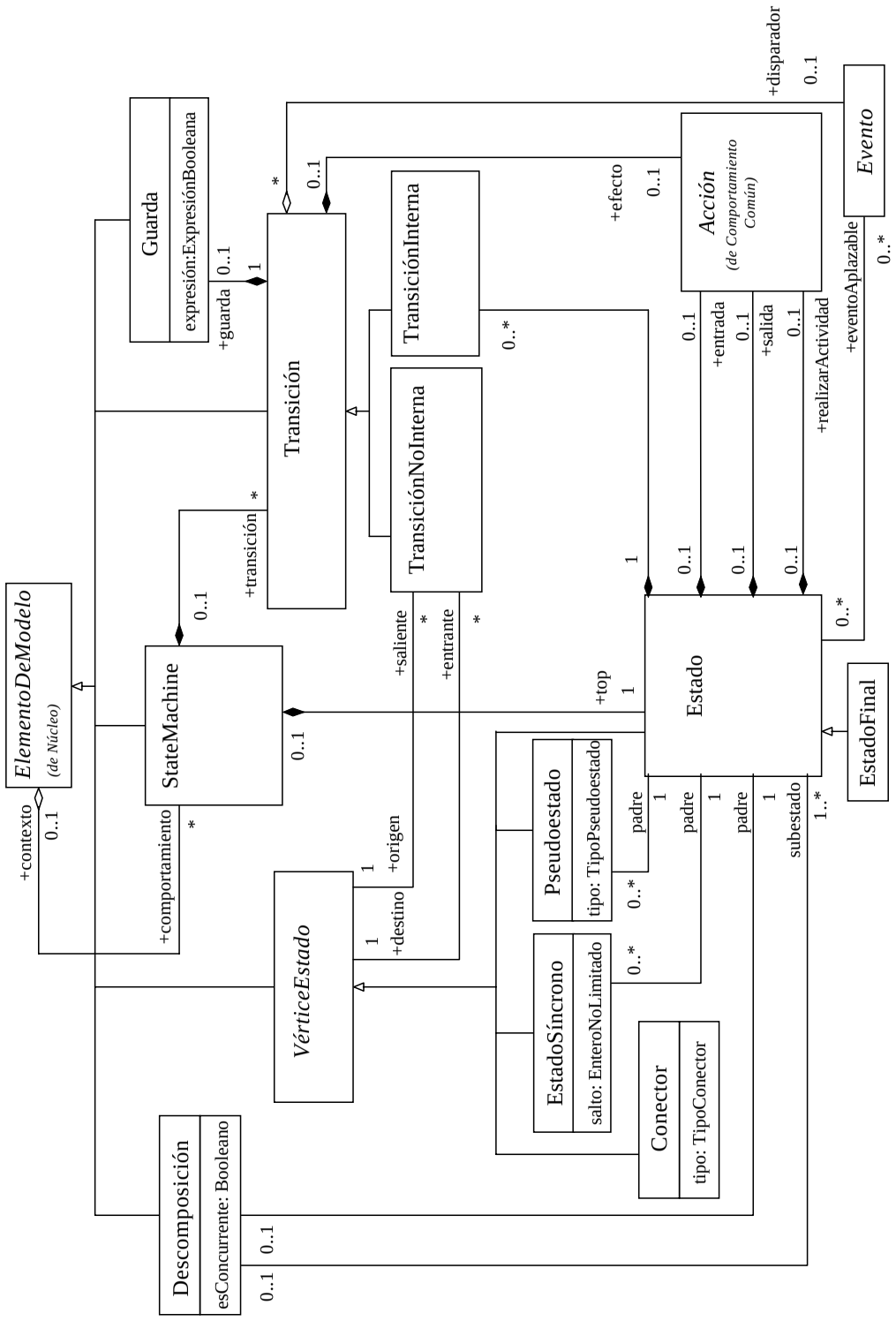


Figura 5.1: Diagrama Estático del metamodelo de UML State Machines

Una de las principales diferencias que podemos encontrar entre el diagrama de clases de la figura 5.1 y el diagrama de clases propuesto en [OMG01b, p. 2–147] es que en nuestra interpretación consideramos que el hecho de que un estado sea simple o compuesto no es una propiedad intrínseca del estado, sino una característica derivada del hecho de si el estado puede ser descompuesto en sub-estados o no. Esta interpretación nos ha llevado a incluir en el Diagrama Estático una nueva clase, de nombre Descomposición que representa la existencia de una relación entre un estado y un conjunto de estados, considerados sub-estados del primero. Una descomposición puede ser de dos tipos, *concurrente* o *no concurrente*. La inclusión de la clase Descomposición, unida al hecho de que en nuestro metamodelo no consideramos la noción de *submachine state* (puesto que según se afirma en [OMG01b, p. 2–154], «un submachine state es una conveniencia sintáctica [...] y es semánticamente equivalente a un estado compuesto») provoca que en el Diagrama Estático la clase Estado tenga sólo una sub-clase.

Nuestra propuesta resuelve un problema potencial relacionado con los estados simples y compuestos. Durante el proceso de construcción de un state machine, puede ocurrir que un estado que inicialmente se considere que es un estado simple pase a ser un estado compuesto. Según la especificación oficial de State Machines [OMG01b, p. 2–147], la situación descrita conllevaría que una instancia de la clase EstadoSimple se convirtiera en una instancia de la clase EstadoCompuesto. Esta situación de cambio de clase para una instancia concreta ha sido analizado en la literatura con el nombre *anomalía de reclasificación de objeto* [CZ97], y considerado como problemático durante el análisis y diseño orientado a objeto. Observemos que en la versión que proponemos en este trabajo se evita la aparición de este problema. En efecto, para cambiar el rol de un estado simple a estado compuesto basta con añadir una instancia adecuada de la clase Descomposición, sin que sea necesario reclasificar ningún objeto. Otra posible solución para evitar el problema de la reclasificación de objetos está basada en la consideración de una única clase Estado, de tal forma que las instancias de dicha clase puedan estar calificadas (a través de un atributo) como simples o como compuestas. Esta solución conllevaría la introducción de una relación recursiva de la clase Estado a sí misma para poder representar la jerarquía de estados dentro de un state machine. Sin embargo, tal y como se afirma en [Lee99] en el contexto del modelo E/R, «la semántica de las relaciones recursivas es bastante difícil captar». Por ello la solución que hemos decidido adoptar es la que hemos

expuesto anteriormente y que incluye una clase Descomposición, ya que creemos que esta solución puede mejorar la comprensibilidad del metamodelo.

Otra modificación que proponemos está relacionada con la clasificación de los diferentes tipos de vértices de estado. En particular, hemos creído conveniente diferenciar entre, por una parte, los vértices de tipo inicial, historia superficial e historia profunda (a los que seguimos considerando como Pseudoestados), y por otra parte, los vértices de tipo unión , bifurcación y cruce, a los que pasamos a denominar Conectores. La razón que nos ha motivado a esta distinción es que, mientras que los vértices de tipo inicial e historia han de estar asociados a un estado (que desempeña el papel de *padre* del vértice), los vértices de tipo unión, bifurcación y cruce pueden no estar asociados a ningún estado. Además, a diferencia de los primeros, el papel de estos tres últimos tipos de vértices está íntimamente ligado a la definición de *transiciones compuestas*, siendo una parte fundamental de la definición de este tipo de transiciones, sirviendo como conectores (de ahí el nombre) de transiciones simples. Obviamente, cuando se realiza una representación gráfica de un state machine, cualquier vértice de tipo unión, bifurcación o cruce aparecerá dibujado en el interior de un estado, pero qué estado en concreto sea éste no tiene mayor relevancia para el modelo. En este sentido compartimos el punto de vista de otros autores de formalizaciones del formalismo Statecharts, puesto que ellos tampoco asocian estados padre a este tipo de vértices (véase por ejemplo [HN96]). Una desventaja de el planteamiento que desdobla la clase original (Pseudoestado) en dos (Pseudoestado y Conector) es que se incrementa el número de clases en el metamodelo, y por tanto se incrementa la complejidad de éste. En este caso concreto, entendemos que la distinción entre Pseudoestado y Conector es de naturaleza intrínsecamente conceptual, y por tanto hemos considerado esencial el incluirla para conseguir una mejor comprensión del lenguaje State Machines.

El último cambio que proponemos es la especialización de la clase Transición en dos sub-clases, de tal forma que sea posible distinguir si una transición es una transición interna asociada a un estado o no. Nuestra intención es la de remarcar el hecho de que el origen y el destino de una transición interna son siempre el mismo y que además este origen y destino sólo puede ser un estado (el mismo estado, por tanto). En particular el origen y destino de una transición interna no puede ser ni un pseudoestado ni un conector ni un estado síncrono.

Por último, la semántica estática del nuevo diagrama de clases UML propuesto se

describe por medio de expresiones OCL. Básicamente éstas son las mismas reglas que se proponen en [OMG01b], si bien añadiendo algunas reglas adicionales. Estas reglas están en sintonía con las restricciones impuestas para el metamodelo NÓESIS de Statecharts que hemos mostrado en el capítulo anterior, y por tanto no suponen una contribución original especial para el metamodelo UML, de ahí que no las incluyamos aquí de forma explícita.

5.2.2. Nivel de Status

La semántica dinámica (semántica de ejecución) de State Machines está relacionada con el Nivel de Status de la arquitectura NÓESIS ya que este nivel recoge los aspectos relativos al status de un state machine en un momento dado. Para especificar la semántica dinámica de State Machines hemos tenido en cuenta el extracto que afirma que el comportamiento real de un statechart «consiste de una serie de instantáneas detalladas [...]. La primera de la secuencia es el status inicial, y cada una de las instantáneas siguientes se obtiene de su predecesora por medio de la ejecución de un paso» [HN96]. Si bien esta cita ha sido entresacada de la definición de la semántica de Statecharts para una herramienta CASE particular (la herramienta STATEMATE, descrita en [HP98]), su contenido puede ser considerado como una descripción elemental, ampliamente aceptada, del comportamiento de un statechart. Además, es importante tener en cuenta que el Documento de Especificación de UML la semántica dinámica está descrita en lenguaje natural (inglés), y que en estas descripciones textuales son introducidos diferentes conceptos imprescindibles para analizar el comportamiento, como por ejemplo la noción de *configuración de estado activo*.

De la misma forma que los conceptos estáticos se representan en un diagrama de clases UML, nuestra propuesta defiende adoptar un enfoque análogo para la representación de los conceptos dinámicos. Así, proponemos un diagrama de clases, denominado Diagrama Dinámico (figura 5.2), que recoge conceptos utilizados específicamente para la descripción de la semántica dinámica. Es necesario advertir que hemos necesitado incluir una innovación en la notación estándar de UML para la representación del Diagrama Dinámico. Obviamente, todos los conceptos especificados en el Diagrama Estático de la figura 5.1 son necesarios para la correcta descripción de la dinámica, y por tanto también han de ser incluidos en el Diagrama

Dinámico. Pero ocurre que muchos de estos conceptos aparecen tal cual en ambos diagramas, aunque algunos otros han sido ligeramente modificados (por la adición de nuevos atributos, métodos o relaciones entre clases) al aparecer en el Diagrama Dinámico. Para incrementar la legibilidad simultánea y la comparación de ambos diagramas, en el Diagrama Dinámico hemos dibujado en gris aquellas clases y elementos que permanecen invariables con respecto al Diagrama Estático. Esta es una posibilidad no recogida en la notación UML estándar, ya que en la descripción de UML no se analizan problemas de comparación o transformación entre diagramas de clases, por lo que una notación tal no resulta necesaria. Al introducir una noción de comparación entre diagramas de clases, como en nuestro caso, aparece de manera natural la conveniencia de disponer de una notación que facilite dicha comparación.

Así pues, el Diagrama Dinámico pertenece al Nivel de Status y recoge, junto con los conceptos que se consideran independientes del status, aquellos conceptos que son necesarios para determinar el status de un state machine en un momento dado. En particular, cada una de las 'instantáneas' de la secuencia que se nombraban en la anterior cita literal corresponden con un modelo del Diagrama Dinámico que proponemos. En otras palabras, una 'instantánea' puede ser descrita por medio de un Diagrama de Objetos UML que sea instancia del Diagrama Dinámico. La forma en que tal instantánea es obtenida a partir de su predecesora (es decir, la representación de un paso) será analizada en la siguiente sub-sección.

Observemos en el Diagrama Dinámico se ha añadido, respecto al Diagrama Estático, una clase llamada ConfiguraciónEstadoActivo, la cual representa los estados que están activos en un momento dado. Además, en este diagrama es necesario recoger la información relativa a la historia del comportamiento de la state machine. Esto se ha resuelto mediante la incorporación de la información acerca del último sub-estado activo y la configuración básica relativa a un estado. Finalmente, a la clase EstadoSíncrono se le ha asociado un nuevo atributo, que representa, para cada objeto de la clase, la diferencia entre el número de veces que sus transiciones entrantes y salientes son disparadas.

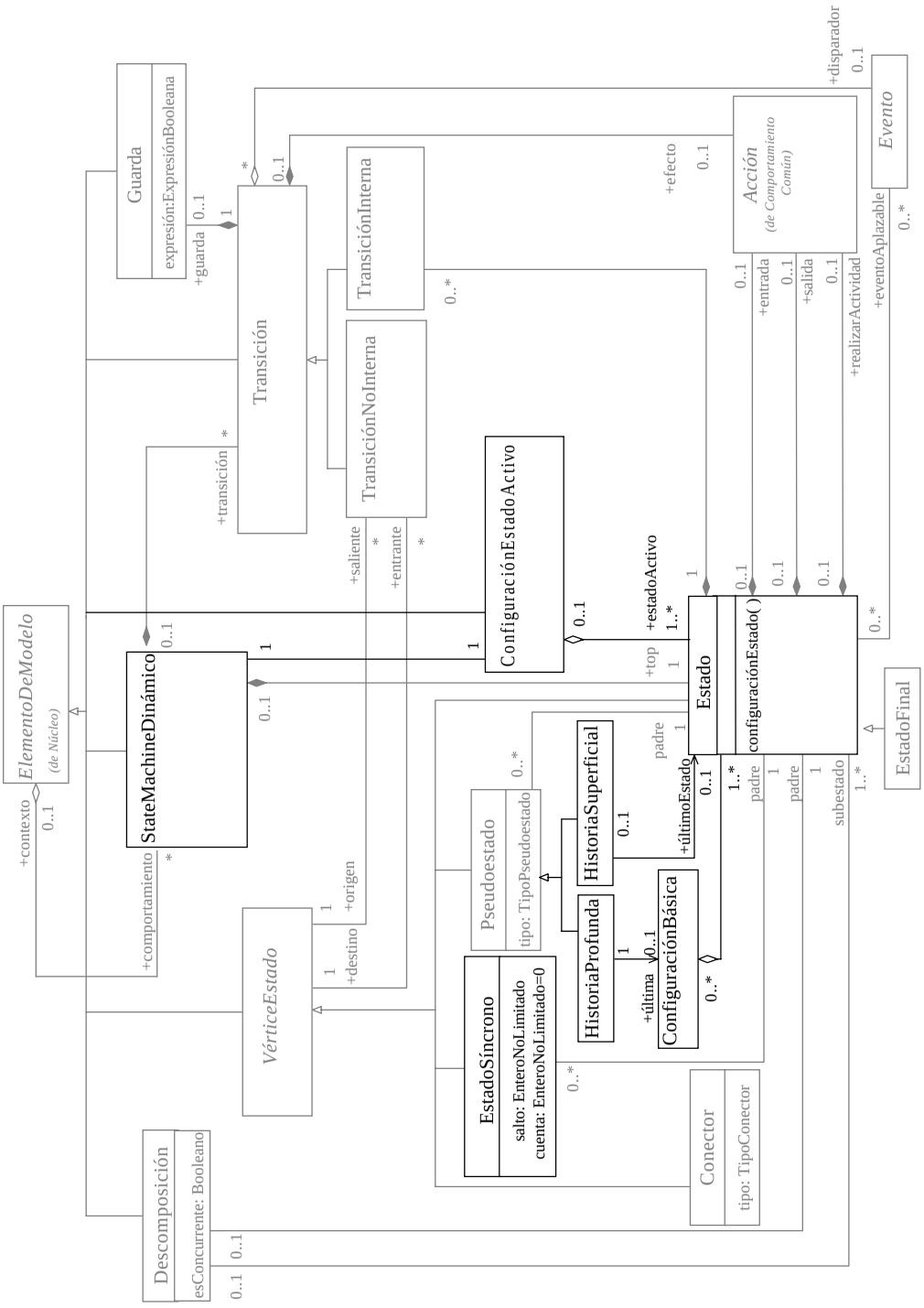


Figura 5.2: Diagrama Dinámico del metamodelo de UML State Machines

5.2.3. Transformaciones T_0 y T

Una vez que los Diagramas Estático y Dinámico han sido determinados, el metamodelo ha de ser completado, siguiendo las directrices de la arquitectura NÓESIS, con dos transformaciones. Por una parte hay que determinar el procedimiento que debe seguirse para calcular el status inicial. Este procedimiento, representado en el metamodelo por medio de una transformación T_0, calcula un *state machine dinámico* (es decir, un modelo del Diagrama Dinámico) a partir de un *state machine estático* (que es, a su vez, un modelo del Diagrama Estático). Por otra parte es necesario definir el procedimiento que debe seguirse para calcular, a partir de un state machine dinámico que represente el status actual, otro state machine dinámico que represente el siguiente status. Este segundo procedimiento, representado en el metamodelo por medio de una transformación T, recoge el contenido de lo que significa un paso *run–to–completion*. Además este procedimiento permite la construcción de una secuencia de modelos del Diagrama Dinámico, secuencia que representa una traza de ejecución de un state machine.

En concreto nuestra propuesta incluye la consideración de una noción de transformación entre diagramas de clases para formalizar los dos procedimientos descritos. Como ya hemos observado, ni en la definición actual de UML ni en la de MOF se incluye noción alguna de transformación entre diagramas de clases. Por tanto hemos tenido que recurrir a otros ámbitos que tratan el tema de la (meta)modelización para encontrar una solución al problema. En particular, en el ámbito de la ingeniería del método existen distintos enfoques que utilizan diversas nociones de transformación para resolver problemas similares. Puesto que nos estamos enfrentando a un problema muy concreto, nos conformaremos aquí con proporcionar una definición de transformación muy general que nos sirva como marco para presentar la solución a nuestro caso particular. Más específicamente, dados dos diagramas de clases C y C' pertenecientes al Nivel de Metamodelo (dentro de la Arquitectura de cuatro niveles, véase la figura 1.1 en la página 19–Capítulo 1) definimos una *transformación de C en C'* como un método que permite determinar un modelo de C' a partir de un modelo de C. Aplicando esta definición genérica al caso particular de las transformaciones T_0 y T que acabamos de describir de manera informal, hemos de hacer explícito qué diagramas de clases están involucrados en cada una de las transformaciones. En concreto, T_0 será una transformación trazada desde el Diagrama Estático has-

Tabla 5.1: Transformaciones del metamodelo de UML State Machines

$T_0(M$: StateMachine) $\Rightarrow M_D^0$: StateMachineDinámico
crear M_D^0 como copia de M
con M_D^0 hacer
configuraciónEstadoActivo.estadoActivo ← top.configuraciónEstado()

$T_1(M_D$: StateMachineDinámico, eventoActual:Evento) $\Rightarrow M_1$:StateMachineIntermedio1
crear M_1 como copia de M_D
con M_1 hacer
transiciónCompuestaDisparable ←
calculaDisparables(eventoActual, M_D.configuraciónEstadoActivo)

$T_2(M_1$: StateMachineIntermedio1) $\Rightarrow M_2$: StateMachineIntermedio2
crear M_2 como copia de M_1
con M_2 hacer
paso ← calculaPaso(M_1.transiciónCompuestaDisparable)

$T_3(M_2$: StateMachineIntermedio2) $\Rightarrow M_D^{'}$: StateMachineDinámico
crear $M_D^{'}$ como copia de M_2
con $M_D^{'}$ hacer
M_2.paso.ejecutar()

ta el Diagrama Dinámico y T será una transformación trazada desde el Diagrama Dinámico hasta el propio Diagrama Dinámico.

Para la descripción de las transformaciones hemos adoptado la siguiente sintaxis concreta. La especificación de cada transformación contiene, en primer lugar, su signatura, que especifica el nombre de la transformación, sus parámetros y la especificación del resultado que se obtiene tras la aplicación de la transformación. Uno de los parámetros determina el modelo que será modificado, y el resto determina la información específica que cada transformación tiene que conocer para llevar a cabo su función. Además, cada transformación se acompaña de un algoritmo esquemático que describe su funcionalidad. Una vez descrita la especificación textual que vamos a utilizar, vamos a destacar los aspectos más fundamentales de cada una de las

transformaciones.

La determinación del status inicial de un state machine se modeliza en nuestro metamodelo por medio de la transformación T_0 (véase la tabla 5.1), que a partir de un state machine estático M obtiene un state machine dinámico M_D^0. La transformación T_0 no incluye en M_D^0 ninguna información relativa a la historia del state machine puesto que este modelo representa el primer status. Por su parte, la configuración inicial de estados activos se obtiene calculando la configuración de estados por defecto para el estado raíz –top–, mediante la utilización de la operación configuraciónEstado de la clase Estado (véanse la figura 5.2 y la tabla 5.1). Hemos de observar que la transformación T_0 debería ser analizada con un mayor grado de detalle, utilizando la noción de *refinamiento* que introduce la arquitectura NÓESIS. Sin embargo ha de tenerse en cuenta que en el Documento de Especificación de UML [OMG01b]la determinación del status inicial para un state machine es algo que tampoco se analiza con detenimiento.

En segundo lugar, la noción de paso *run–to–completion* se modeliza mediante la transformación T, que a partir de un state machine dinámico M_D obtiene otro state machine dinámico M'_D. La descripción detallada y precisa de un paso *run–to–completion* es una tarea enormemente compleja, que es probablemente imposible de realizar de forma directa. Nos hemos inspirado en la descripción algorítmica de [HN96], referencia en la que se especifican tres sub-tareas (que, a su vez, no son en absoluto triviales) para definir un paso. Esta idea nos ha llevado a considerar un refinamiento en dos fases de la transformación T. Estas fases se representan en el metamodelo mediante dos diagramas de clases adicionales (figuras 5.3 y 5.4), en los que hemos empleado la misma notación que utiliza colores que ya hemos explicado anteriormente (de tal forma que los elementos de cada diagrama de clases que permanecen inalterados con respecto al diagrama de clases anterior aparecen dibujados en gris). Como consecuencia de la especificación de ambos diagramas de clases, en el metamodelo es necesario incluir tres transformaciones intermedias, referidas por T_i $(i = 1, 2, 3)$. Para evitar malentendidos hay que aclarar que estas transformaciones no tienen correspondencia directa con las tres sub-tareas descritas en [HN96] y que hemos citado anteriormente. La razón es que nuestro metamodelo describe el sub-lenguaje UML State Machines mientras que la referencia [HN96] maneja otra versión distinta de Statecharts. Vamos a describir a continuación brevemente las tres transformaciones intermedias (T_i). En concreto, la transformación T_1 calcula

las transiciones compuestas disparables –enabled compound transitions, ECT's– a partir de la instancia actual de eventos, la configuración de estados activa, los distintos conectores (utilizados para calcular las transiciones compuestas), los valores de variables y guardas, etc. Esta tarea es realizada por la operación calculaDisparables de la clase StateMachineIntermedio1 (véanse la figura 5.3 y la tabla 5.1). Un paso se calcula por medio de la transformación T_2 a partir del conjunto de ECT´s, resolviendo los posibles conflictos entre transiciones de este conjunto y teniendo en cuenta las prioridades de disparo. En este caso se utiliza la operación calculaPaso de la clase StateMachineIntermedio2 (figura 5.4 y la tabla 5.1). Finalmente, el paso se ejecuta por medio de la transformación T_3, provocando ésta la ejecución de ciertas acciones y dando lugar a un nuevo state machine dinámico, el cual representa el nuevo status. Esta tarea es llevada a cabo por la operación ejecutar de la clase Paso (figura 5.4 y la tabla 5.1). Es necesario observar que cada una de estas transformaciones podrían ser, a su vez, refinadas, añadiendo nuevas fases y transformaciones hasta que se alcance el grado de detalle deseado por parte del meta–analista. Más aún, cada una de las operaciones utilizadas en estas transformaciones debería estar implementada a través de un método concreto. A este respecto es interesante mencionar el algoritmo detallado que se muestra en [LP99a], algoritmo que especifica un paso *run–to–completion*.

5.3. Trabajos relacionados y cuestiones abiertas

Tal y como hemos comentado en varias ocasiones en esta memoria existen distintos trabajos en la literatura en los que se definen semánticas formales para el formalismo Statecharts original de Harel (por ejemplo [Har87, HRdR92, vdB94, HKCK95, EGKP97, HP98]). Sin embargo, existe un número menor de investigaciones que traten la formalización de la semántica de UML State Machines. Además, tal y como se afirma en [Kwo00], algunos de estos trabajos (como por ejemplo [GLM99, LMM99, EHHS00]) no proporcionan una formalización completa de State Machines, sino que consideran sólo algunas construcciones básicas.

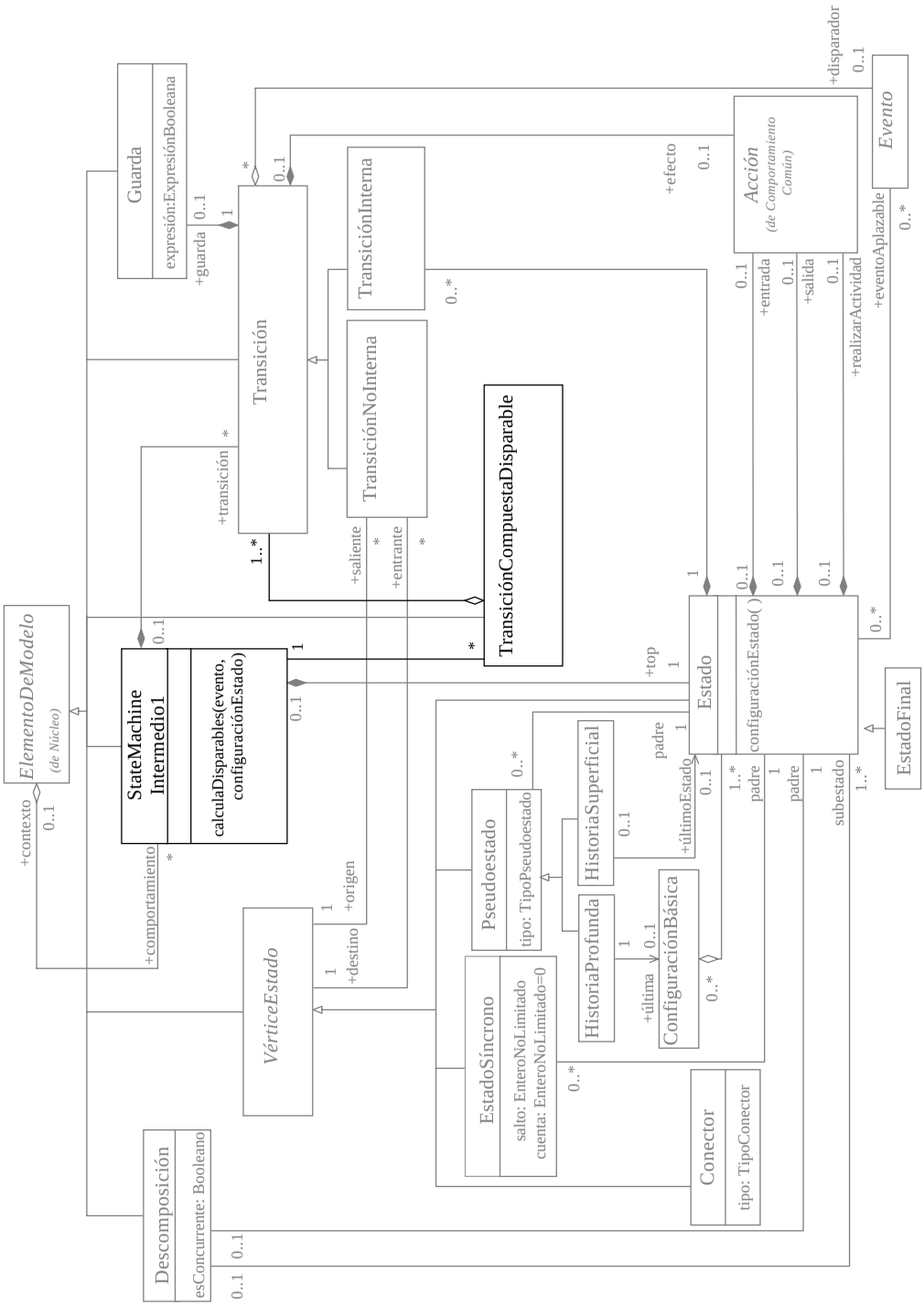


Figura 5.3: Primer Diagrama Intermedio del metamodelo de UML State Machines

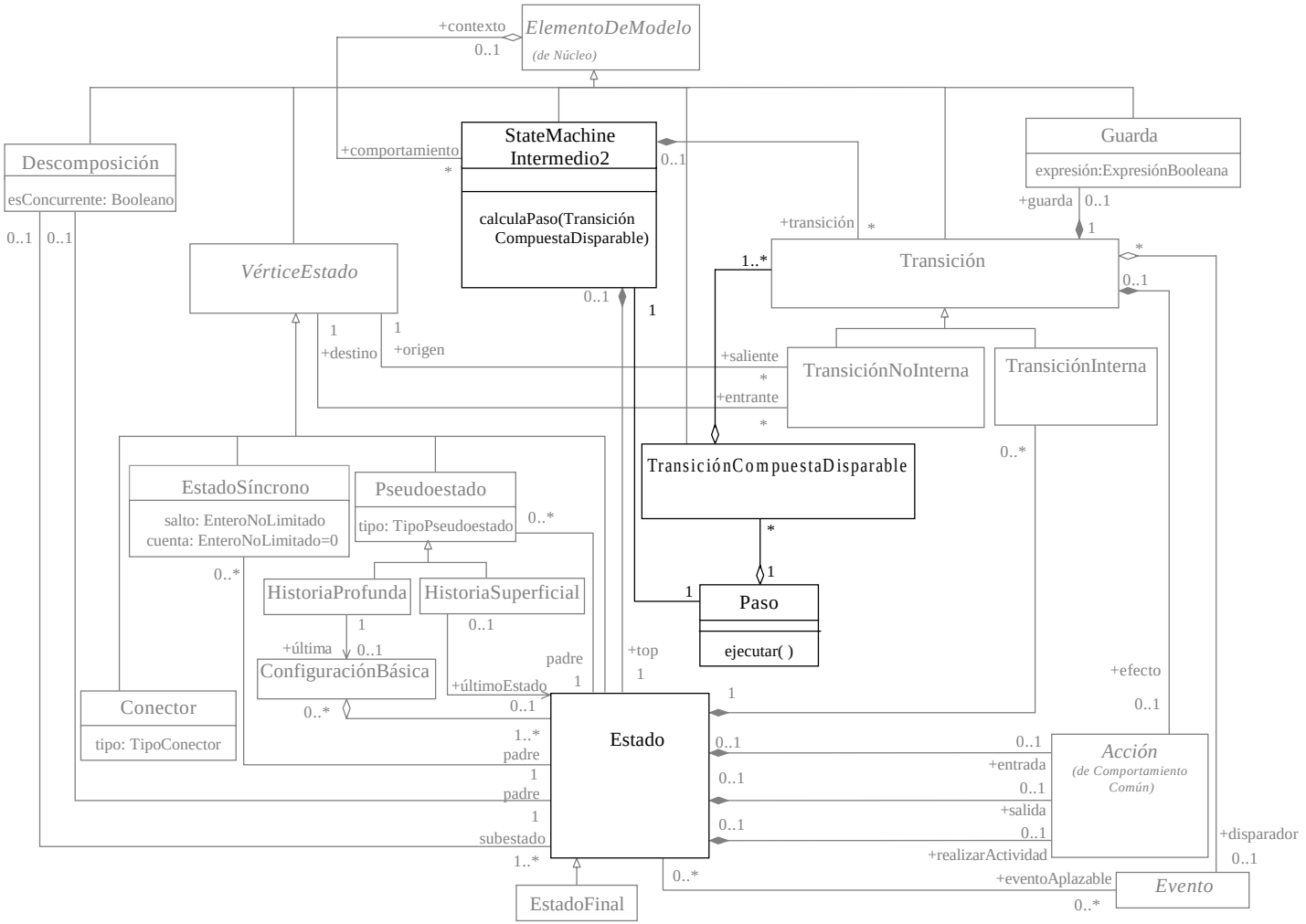


Figura 5.4: Segundo Diagrama Intermedio del metamodelo de UML State Machines

En general, al tratar la formalización de State Machines, los autores proponen utilizar diversas notaciones formales tales como Reglas de Reescritura –Rewrite Rules– [LP99a, Kwo00], Autómatas Jerárquicos –Hierarchical Automata– [LMM99], Máquinas de Estado Abstractas –Abstract State Machines– [BCR00] o Object-Z [MK98]. Sin embargo, del mismo modo que los autores de [EHHS00], creemos que estos enfoques tienen la importante desventaja de ser difíciles de leer y entender, y que por lo tanto no son lo más adecuados para explicar la semántica a usuarios que no estén familiarizados con notaciones formales.

De ahí que nuestra propuesta, sin negar la utilidad de las notaciones formales en determinados contextos, sea la de adoptar una perspectiva de metamodelización. Los autores de [EHHS00] plantean también una perspectiva de metamodelización para representar la semántica dinámica de State Machines. En concreto, estos autores proponer extender el metamodelo de UML State Machines con información sobre estados, que considerada bajo el prisma de la arquitectura NÓESIS puede entenderse como una ampliación del metamodelo que incluya información dependiente del status (perteneciente al Nivel de Status). Es por tanto un enfoque similar al nuestro, pero a diferencia de nuestro metamodelo, el propuesto por Engels et al. no cubre todas las características de State Machines, de tal forma que el metamodelo dinámico propuesto es incompleto. Exceptuando este aspecto compartido, los restantes aspectos de ambos enfoques son bastante diferentes, principalmente porque ellos adoptan una forma diferente de metamodelizar el paso *run–to–completion*: ellos utilizan diagramas de colaboración y nosotros usamos la noción de transformación.

Aunque en efecto la noción de transformación ha sido defendida por distintos autores [Ver93, MHR96, tHV97, DZ00, DZE00] como un artefacto necesario para resolver problemas de metamodelización similares al que aquí planteamos, no es sin duda la única noción posible. Otros artefactos distintos han sido propuestos en la literatura como medio para especificar la secuencia de pasos de un procedimiento. Por ejemplo, el valor de la *modelización de procesos –process modeling–* para representar de forma rigurosa y explícita este tipo de aspectos funcionales ha sido probado por ejemplo en [SO94]. La realización de un análisis detallado y en profundidad de cuál es el artefacto más adecuado para representar los procedimientos de cálculo del status inicial y el siguiente status dentro del contexto de la representación del comportamiento aparece como un problema abierto.

Esta investigación plantea otros interesantes problemas abiertos. Observemos

que en particular nuestro trabajo se centra en el análisis de la semántica dinámica de un único state machine. Sin embargo, una interesante línea de investigación supone estudiar como pueden recogerse dentro de nuestra propuesta las interacciones entre diferentes state machines (por ejemplo, las relaciones entre los state machines asociados a instancias de distintas clases en un Diagrama de Objetos UML). Otras líneas de trabajo abiertas tienen que ver por una parte con aspectos relativos al refinamiento de state machines, tales como subtipos, herencia estricta y refinamiento general, y por otra con el análisis de cuáles pueden ser las relaciones de la noción UML de estereotipo con nuestro enfoque.

Conclusiones

En este trabajo hemos presentado una arquitectura, denominada arquitectura NÓESIS, cuyo objetivo es el de servir como marco de trabajo genérico para la representación de aspectos de comportamiento durante el desarrollo de sistemas software, en particular sistemas de información. La principal aportación de esta arquitectura es que proporciona un enfoque que distingue, a la hora de analizar los aspectos de comportamiento, aquellas características que no dependen de la situación concreta en que se encuentre el sistema modelizado de aquellas otras que sí están en función de cada situación concreta. En este trabajo hemos hecho referencia a estos dos aspectos diferenciados mediante los términos *Vista Estática* y *Vista Dinámica*, y en concreto la arquitectura NÓESIS recoge estos aspectos mediante la introducción de dos niveles, el *Nivel de Independencia de Status* y el *Nivel de Status*, y dos transformaciones genéricas, denotadas por T_0 y T. Estas dos transformaciones representan, respectivamente, la necesidad del establecimiento de la situación inicial y la noción de movimiento de una situación a otra, nociones ambas que pertenecen a la esencia del significado de 'comportamiento'.

Una de las características fundamentales de esta arquitectura es que establece un marco genérico, no ligado ni a ningún nivel de abstracción concreto ni a ninguna metodología de diseño ni lenguaje de modelización particulares. Distintos tipos de usuarios implicados en el desarrollo de sistemas de información, tales como ingenieros del software, desarrolladores de herramientas CASE o ingenieros del método, pueden beneficiarse del uso de la arquitectura en función de sus necesidades particulares. Un analista o diseñador de sistemas podrá usar este enfoque de comprensión del comportamiento para obtener una interpretación más precisa del sistema que esté siendo modelizado. En particular, esta tarea de modelización será más fácil si el lenguaje utilizado por el ingeniero del software para modelizar las características de comportamiento ha sido descrito de forma precisa siguiendo las directrices de la

arquitectura NÓESIS. A su vez, estos lenguajes para la modelización de comportamiento pueden formar parte de herramientas CASE, herramientas que sin duda son de gran utilidad para el desarrollo de software. La consideración de nuestra arquitectura puede proporcionar soporte para el análisis de herramientas CASE existentes y para el desarrollo de otras nuevas, siempre en función de las intenciones de los desarrolladores de estos programas. Por último, los ingenieros del método pueden utilizar la arquitectura como marco para el estudio de distintos formalismos de comportamiento, siendo este un primer paso para el análisis posterior de tareas propias de la ingeniería del método, tales como la comparación, transformación y adaptación de métodos.

El marco genérico que proporciona la arquitectura ha sido aprovechado en esta memoria en dos sentidos diferentes. En primer lugar hemos presentado un metamodelo del formalismo Statecharts. Para la realización de este metamodelo hemos tenido en cuenta por una parte la arquitectura, y por otra hemos utilizado como técnica de metamodelización la técnica NÓESIS. La presentación de esta técnica se ha ilustrado además con un ejemplo detallado de un metamodelo del modelo de bases de datos RM/T. Por otra parte en esta memoria hemos presentado un metamodelo de la versión de Statecharts que bajo el nombre State Machines se ha incluido en la especificación del lenguaje UML. Como prueba de la versatilidad de la arquitectura NÓESIS, ésta ha sido usada también como marco en este segundo metamodelo, si bien utilizando en este caso como técnica de metamodelización el propio lenguaje UML.

En este trabajo se han destacado además distintas posibles líneas de trabajo futuro. En primer lugar, y con respecto al desarrollo de la técnica NÓESIS, puede ser interesante investigar cómo podría automatizarse la aplicación de la técnica mediante el uso de herramientas metaCASE (ya sean existentes o desarrolladas de forma específica). La utilización de este tipo de herramientas está ligada a la especificación de una semántica simbólica para la técnica NÓESIS. Recordemos que la semántica que hemos presentado aquí está orientada al usuario humano, y que esta semántica no es apropiada para su representación en un ordenador. La investigación sobre la especificación de lenguajes dotados de dos semánticas (una orientada al usuario humano y otra orientada a su implementación en el ordenador), a los que podríamos denominar *lenguajes bisemánticos*, se presenta como una interesante línea de trabajo. Otro aspecto ligado al desarrollo de la técnica NÓESIS en el que

puede ser interesante profundizar es en la relación de la técnica con la noción de transformación. En este trabajo hemos manipulado transformaciones entre modelos, modelos que se obtenían como instancias de un metamodelo. Por su parte la última versión de la técnica NÓESIS incluye nociones de transformaciones de metamodelos, en concreto de transformaciones de soportes. Será necesario investigar las posibles relaciones existentes entre transformaciones entre metamodelos y transformaciones entre modelos que sean instancias de metamodelos. Además, otros aspectos de las transformaciones, más ligados al tipo de sintaxis concreta que conviene utilizar (por ejemplo si es posible utilizar una sintaxis del estilo de OCL) o a la manera de dotar de semántica a estas transformaciones (por ejemplo utilizando como soporte la teoría de las transformaciones de grafos) pueden ser investigados. Por último y con respecto a la profundización en el estudio de los metamodelos concretos que hemos presentado, destacaremos las múltiples posibilidades que ofrece el metamodelo de UML State Machines como fuente de líneas de trabajo futuras. Tal y como hemos observado en el capítulo correspondiente, nuestro trabajo se ha centrado en el análisis de las nociones de comportamiento relativas a un único state machine. Sin embargo, una interesante línea de investigación supone estudiar como pueden recogerse dentro de nuestra propuesta las interacciones entre diferentes state machines (por ejemplo, las relaciones entre los state machines asociados a instancias de distintas clases en un Diagrama de Objetos UML). Otras líneas de trabajo abiertas tienen que ver por una parte con aspectos relativos al refinamiento de state machines, tales como subtipos, herencia estricta y refinamiento general, y por otra por el análisis de cuales pueden ser las relaciones de la noción UML de estereotipo con nuestro enfoque.

Bibliografía

[ACPT99] Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, and Riccardo Torlone. *Database systems: concepts, languages and architectures*. McGraw Hill, 1999.

[ACS98] Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. Specialization by restriction and schema derivations. *Information Systems*, 23(1):1–38, 1998.

[AES01a] J. Alvarez, A. Evans, and P. Sammut. Mapping between levels in the metamodel architecture. In Gogolla and Kobryn [GK01], pages 34–46.

[AES01b] J. Alvarez, A. Evans, and P. Sammut. MML and the metamodel architecture. Available at http://www.puml.org, 2001.

[AK00] Colin Atkinson and Thomas Kühne. Strict profiles: why and how. In Evans et al. [EKS00], pages 309–322.

[AKHS00] Colin Atkinson, Thomas Kühne, and Brian Henderson-Sellers. To meta or not to meta – that is the question. *Journal of Object Oriented Programming*, pages 32–35, December 2000.

[BCR00] Egon Borger, Alessandra Cavarra, and Elvinia Riccobene. Modeling the dynamics of UML state machines. In Yuri Gurevich, Philipp W. Kutter, Martin Odersky, and Lothar Thiele, editors, *Abstract State Machines 2000*, volume 1912 of *Lecture Notes in Computer Science*, pages 223–241. Springer, 2000.

[Ber91] Claude Berge. *Graphs*. North-Holland, 1991.

[Bez98] J. Bezivin. Who's afraid of ontologies? In *OOPSLA'98 Workshop: Model Engineering, Methods and Tools Integration with CDIF*, October 1998.

[BF94] M. Bonjour and G. Falquet. Concept bases: A support to information systems integration. In G. Wijers, S. Brinkkemper, and T. Wasserman, editors, *Advanced Information Systems Engineering, CAISE'94*, volume 811 of *Lecture Notes in Computer Science*, pages 242–255. Springer, 1994.

[BGGK] Robert Büssow, Robert Geisler, Wolfgang Grieskamp, and Marcus Klar. Integrating Z with dynamic modeling techniques for the specification of reactive systems. Available at http://www.first.gmd.de/~espress/gesamt_public.html.

[BHS01] F. Barbier and B. Henderson-Sellers. The whole–part relationship in object modelling: a definition in cOlOr. *Information and Software Technology*, 43:19–39, 2001.

[BLW96] S. Brinkkemper, K. Lyytinen, and R. J. Welke, editors. *Method Engineering. Principles of method construction and tool suppport.* Proceedings of the IFIP TC8 WG8.1/8.2 Working Conference on Method Engineering, Champan & Hall, 1996.

[BM98] J. Bezivin and P.-A. Muller, editors. *UML '98 – The Unified Modeling Language. Beyond the Notation*, volume 1618 of *Lecture Notes in Computer Science*. Springer, 1998.

[BMW84] A. Borgida, J. Mylopoulos, and H. K. T. Wong. Generalization / specialization as the basis for software specification. In M. Brodie, J. Mylopoulos, and J. Schmidt, editors, *On Conceptual Modeling: Perspectives form Artificial Intelligence, Databases and Programming Languages*. Springer-Verlag, 1984.

[Bra83] R. J. Brachman. What IS–A is and isn't: an analysis of taxonomic links in semantic networks. *Computer*, 16(10):30–36, October 1983.

[Bri96] Sjaak Brinkkemper. Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, 38:275–280, 1996.

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[Bro91] Alan W. Brown, editor. *Integrated Project Support Environments: the Aspect project*, volume 33 of *The APIC Series*. Academic Press, 1991.

[BSH99] Sjaak Brinkkemper, Motoshi Saeki, and A. Frank Harmsen. Meta–modelling based assembly techniques for situational method engineering. *Information Systems*, 24(3):209–228, May 1999.

[CABJ01] William Chan, Richard J. Anderson, Paul Beame, and David H. Jones. Optimizing symbolic model checking for statecharts. *IEEE Transactions on Software Engineering*, 27(2):170–190, 2001.

[CEF+99] Tony Clark, Andy Evans, Robert France, Stuart Kent, and Bernard Rumpe. Response to UML 2.0 Request for information. Available at http://www.puml.org, 1999.

[CEKS01] Tony Clark, Andy Evans, Stuart Kent, and Paul Sammut. The MMF aproach to engineering object-oriented design languages. Available at http://www.puml.org, 2001.

[CG94] S. Cronholm and G. Goldkuhl. Meanings and motives of method customization in CASE environments. In *5th European Workshop on Next Generation of CASE Tools*, Utrecht, The Netherlands, 1994.

[Che76] P. P. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[CKSGS94] Malú Castellanos, Thomas Kudrass, Félix Saltor, and Manuel García-Solaco. Interdatabase existence dependencies: A metaclass approach. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, pages 213–216. IEEE-CS Press, 1994.

[CLL99] Peter Coad, Eric Lefebvre, and Jeff De Luca. *Java modeling in color with UML : enterprise components and process*. Prentice Hall, 1999.

[Cod69] E. F. Codd. Derivability, redundancy and consistency of relations stored in large data banks. Technical Report RJ599, IBM, 1969.

[Cod70] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Cod79] E.F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[Cod90] E.F. Codd. *The relational model for database management, version 2*. Addison Wesley, 1990.

[CZ97] W.W. Chu and G. Zhang. Associations and roles in object–oriented modeling. In D.W. Embley and R.C. Goldstein, editors, *Conceptual Modeling – ER'97*, volume 1331 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 1997.

[Dat85] C. J. Date. *An Introduction to Database Systems*, volume II, chapter 6, pages 241–289. Addison Wesley, 1985.

[Dat92] C.J. Date. *Relational Database Writings: 1989–1991*, chapter 7, pages 285–308. Addison Wesley, 1992.

[Dat99] C. J. Date. Thirty years of Relational: extending the Relational Model. When's an extension not an extension? *Intelligent Enterprise Magazine*, 2(8), June 1999.

[Dat01] C. J. Date. *An Introduction to Database Systems*. Addison Wesley, 7th edition, 2001.

[DB90] James P. Davis and Ronald D. Bonnell. Modeling semantics with concept abstraction in the EARL data model. In F.H. Lochovsky, editor, *Entity-Relationship Approach to Database Design and Querying*, pages 95–110. Elsevier Science Publishers, 1990.

[Dom99] E. Domínguez. *Un Paseo Fenomático*. Academia de Ciencias Exactas, Físicas, Químicas y Naturales de Zaragoza, 1999.

[Dou99] B. Powel Douglas. UML statecharts. *Embeded Systems Programming*, 12(1), Jan 1999.

[Drb99] Milan Drbohlav. Meta-modeling: Theory and practical implications. In *Systems Development Methods for Databases, Enterprise Modeling and Workflow Management*, pages 199–208. Kluwer Academic, 1999.

[DRZ] E. Domínguez, A. L. Rubio, and M. A. Zapata. Dynamic semantics of UML state machines: a metamodeling perspective. To appear in *Journal of Database Management*.

[DRZ00a] E. Domínguez, A. L. Rubio, and M. A. Zapata. Meta–modelling of dynamic aspects: The *Noesis* approach. In J. Bézivin and J. Ernst, editors, *Proceedings of the ECOOP'2000 International Workshop on Model Engineering*, pages 28–35, 2000.

[DRZ00b] E. Domínguez, A. L. Rubio, and M. A. Zapata. A way of dealing with behaviour of state machines. In Reggio et al. [RKR+00], pages 32–37.

[DZ] E. Domínguez and M. A. Zapata. *Noesis: Towards a Situational Method Engineering Technique*. Preprint.

[DZ00] E. Domínguez and M. A. Zapata. Mappings and interoperability: A meta–modelling approach. In T. Yakhno, editor, *Advances In Information Systems – ADVIS 2000*, volume 1909 of *Lecture Notes in Computer Science*, pages 352–362. Springer–Verlag, 2000.

[DZE00] E. Domínguez, M. A. Zapata, and I. Escario. Meta–model transformations as a support for method adaptation. In B. Sanchez, N. Nada, A. Rashid, T. Arndt, and M. Sanchez, editors, *World Multiconference on Systemics, Cybernetics and Informatics – SCI 2000*, volume II Information Systems Development, pages 44–49, 2000.

[DZR97] E. Domínguez, M. A. Zapata, and J. Rubio. A conceptual approach to meta–modelling. In A. Olivé and J.A. Pastor, editors, *Advanced Information Systems Engineering, CAISE'97*, volume 1250 of *Lecture Notes in Computer Science*, pages 319–332. Springer, 1997.

[EGKP97] Hartmut Ehrig, Robert Geisler, Marcus Klar, and Julia Padberg. Horizontal and vertical structuring techniques for statecharts. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR'97 Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 1997.

[EHHS00] Gregor Engels, Jan Hendrik Haussmann, Reiko Heckel, and Stefan Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In Evans et al. [EKS00], pages 323–337.

[EKS00] Andy Evans, Stuart Kent, and Bran Selic, editors. *UML 2000 – The Unified Modeling Language. Advancing the Standard*, volume 1939 of *Lecture Notes in Computer Science*. Springer, 2000.

[EKTW87] J. Eder, G. Kappel, A. M. Tjoa, and R. R. Wagner. Rapid prototyping using the extended relational data model RM/T in a Prolog environment. In *Proceedings of the IASTED International Symposium Applied Informatics*, pages 215, 1–4, 1987.

[EN02] Ramez Elmasri and Shamkant Navathe. *Fundamentals of database systems*. Addison Wesley, 2002.

[EW91] A. N. Earl and R. P. Whittington. *Integrated Project Support Environments: the Aspect project*, chapter 5, pages 71–99. Academic Press, 1991.

[FD96] Donal J. Flynn and Olivia Fragoso Diaz. *Information Modelling: An International Perspective*. Prentice Hall, 1996.

[FH] Kay Fuhrmann and Jan Hiemer. Formal verification of statemate-statecharts. Available at http://www.first.gmd.de/~espress/gesamt_public.html.

[FHL+98] Eckhard D. Falkenberg, Wolfgang Hesse, Paul Lindgreen, Björn E. Nilsson, J.L. Han Oei, Colette Rolland, Ronald K. Stamper, Frans J. M. Van Assche, Alexander A. Verrijn-Stuart, and Klauss Voss. A framework of information systems concepts: The FRISCO report. Technical report,

International Federation for Information Processing, 1998. Available at http://www.liacs.nl/∼verrynst/frisco.html.

[FR99] Robert France and Bernhard Rumpe, editors. *UML '99 – The Unified Modeling Language. Beyond the Standard*, volume 1723 of *Lecture Notes in Computer Science*. Springer, 1999.

[Gei99] Robert Geisler. *Formal Semantics for the Integration of Statecharts and Z in a Metamodel-Based Framework*. PhD thesis, Technischen Universität Berlin, 1999.

[GHD01] Wolfgang Grieskamp, Maritta Heisel, and Heiko Dörr. Specifying embedded systems with statecharts and Z: An agenda for cyclic software components. *Science of Computer Programming*, 40:31–57, 2001.

[GK01] M. Gogolla and C. Kobryn, editors. *UML 2001 – The Unified Modeling Language. Modeling Languages, Concepts, and Tools.*, volume 2185 of *Lecture Notes in Computer Science*. Springer, 2001.

[GKP98] Robert Geisler, Marcus Klar, and Claudia Pons. Dimensions and dichotomy in metamodeling. Technical Report FB 13, No. 98-5, Technical Universitat Berlin, 1998.

[GLM99] S. Gnesi, D. Latella, and M. Massink. Model checking UML statechart diagrams using JACK. In *Proceedings of the IEEE International Symposium on High Assurance Systems Engineering*, 1999.

[Göd31] K. Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme. *Monatshefte für Mathematik und Physik*, XXXVIII:173–198, 1931.

[GV95] J.C. Grundy and J.R. Venable. Providing integrated support for multiple development notations. In J. Iivari and K. Lyytinen, editors, *Advanced Information Systems Engineering, CAiSE'95*, volume 932 of *Lecture Notes in Computer Science*, pages 255–268. Springer, 1995.

[GV96] J. C. Grundy and J. R. Venable. Towards an integrated environment for method engineering. In Brinkkemper et al. [BLW96], pages 45–62.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Har97] A. F. Harmsen. *Situational Method Engineering*. Moret Ernst & Young, 1997.

[HBO94] A. Frank Harmsen, Sjaak Brinkkemper, and Hank Oei. Situational method engineering for information systems project approaches. In A.A. Verrijn-Stuart and T.W. Olle, editors, *Methods and associated tools for the information systems life cycle*, pages 169–194. Elsevier, 1994.

[HG97] D. Harel and E. Gery. Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42, July 1997.

[HK87] Richard Hull and Roger King. Semantic database modeling: survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[HKCK95] Hyoung Seok Hong, Jeong Hyun Kim, Sung Deok Cha, and Yong Rae Kwon. Static semantics and priority schemes for statecharts. In *9th Annual International Computer Software and Applications Conference COMPSAC'95*, pages 114–120. IEEE Computer Society Press, 1995.

[HLN+90] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, Apr 1990.

[HN96] David Harel and Amnon Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, Oct 1996.

[Hoa73] C.A.R. Hoare. Hints on programming language design. In *ACM Principles of Programming Languages*. ACM Press, 1973.

[HOT76] P. Hall, J. Owlett, and S. Todd. Relations and entities. In G. M. Nijssen, editor, *Modelling in Data Base Management Systems*. North Holland, 1976.

[HP98] David Harel and Michal Politi. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw-Hill, 1998.

[HPSS87] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman. On the formal semantics of statecharts. In *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, pages 54–64. IEEE Press, 1987.

[HR96] T. Halpin and P. Ritson. Entity integrity revisited. *The Australian Computer Journal*, 28(3):73–80, August 1996.

[HR00] David Harel and Bernhard Rumpe. Modeling languages: Syntax, semantics and all that stuff. Part I: The basic stuff. Technical report, The Weizmann Institute of Science, Rehovot, Israel, MCS00-16, 2000. Available at http://www4.informatik.tu-muenchen.de/papers/HR00.html.

[HRdR92] J.J. Hooman, S. Ramesh, and W.P. de Roever. A compositional axiomatization of statecharts. *Theoretical Computer Science*, 101:289–335, 1992.

[HS96] F. Harmsen and M. Saeki. Comparison of four method engineering languages. In Brinkkemper et al. [BLW96], pages 209–231.

[HSB97] B. Henderson-Sellers and A. Bulthuis. *Object–oriented Metamethods*. Springer, 1997.

[HV95] Tom Holvoet and Pierre Verbaeten. Petri charts: An alternative technique for hierarchical net construction. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2688–2693. IEEE, 1995.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1999.

[KCL87] M. Kao, N. Cercone, and W. Luk. What do you mean 'null'?: turning null responses into quality responses. In *Proceedings of the Third International Conference on Data Engineering*, pages XIV+666, 356–364, 1987.

[KER99] S. Kent, A. Evans, and B. Rumpe. UML semantics FAQ. Available at http://www.cs.ukc.ac.uk/pubs/1999/977/, 1999.

[KLR96] S. Kelly, K. Lyytinen, and M. Rossi. Metaedit+: A fully configurable multi–user and multi–tool CASEand CAME environment. In P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, editors, *Advanced Information Systems Engineering, CAISE'96*, volume 1080 of *Lecture notes in Computer Science*, pages 1–21. Springer, 1996.

[Kwo00] Gihwon Kwon. Rewrite rules and operational semantics for model checking UML statecharts. In Evans et al. [EKS00], pages 528–540.

[LdBC00] Gerald Lüttgen, Michael Von der Beeck, and Rance Cleaveland. A compositional approach to statecharts semantics. *ACM Software engineering notes*, 25(6):120–129, 2000.

[Lee99] H.-K. Lee. Semantics of recursive relationships in entity-relationship model. *Information and Software Technology*, 41(13):877–886, September 1999.

[Lev90] F. J. Leven. Databank design–modelling with data. *Software Kurier für Mediziner und Psychologen*, 3(1):1–13, 1990.

[Llo98] Jorge Lloret. *Un modelo para el diseño conceptual de base de datos: el modelo EAN*. PhD thesis, Universidad de Zaragoza, 1998.

[LMM99] Diego Latella, Istvan Majzik, and Mieke Massink. Towards a formal operational semantics of UML statechart diagrams. In *3rd International Conference on Formal Methods for Open Object-Oriented Distributed Systems*, pages 331–344. Kluwer Academic Publishers, 1999.

[LP91] Wilf Lalonde and John Pugh. Subclassing $\neq$ subtyping $\neq$ is–a. *Journal of Object Oriented Programming*, pages 57–62, January 1991.

[LP99a] Johan Lilius and Ivan Porres Paltor. Formalising UML state machines for model checking. In France and Rumpe [FR99], pages 430–445.

[LP99b] Johan Lilius and Ivan Porres Paltor. The semantics of UML state machines. Technical Report 273, Turku Centre for Computer Science, May 1999.

[MBJK90] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, 8(4):325–362, October 1990.

[MHR96] P. Marttiin, F. Harmsen, and M. Rossi. A functional framework for evaluating method engineering environments: The case of maestro II/Decamerone and MetaEdit+. In Brinkkemper et al. [BLW96], pages 63–86.

[MK98] S. Mann and M. Klar. A metamodel for object-oriented statecharts. In *The Second Workshop on Rigorous Object-Oriented Methods, ROOM 2*, May 1998.

[Moi88] Toril Moi. *Teoría Literaria Feminista*. Cátedra, 1988.

[MPBE95] Hafedh Mili, Francois Pachet, Ilham Benyahia, and Fred Eddy. Metamodeling in OO: OOPSLA'95 workshop summary. In *Addendum to the Proceedings of the 10th OOPSLA Conference*, pages 105–110. ACM Press, 1995.

[MSPT96] Andrea Maggiolo-Schettini, Adriano Peron, and Simone Tini. Equivalence of statecharts. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR'96 Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*. Springer, 1996.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[NBY96] C. Nicolle, D. Benslimane, and K. Yentongnon. Multi–data models translations in interoperable information systems. In J. Mylopoulos and Y. Vassiliou, editors, *Advanced Information Systems Engineering,*

CAiSE'96, volume 1080 of *Lecture Notes in Computer Science*, pages 176–192. Springer, 1996.

[Nis99] Nimal Nissake. *Formal Specification. Techniques and applications.* Springer, 1999.

[NSKL99] G. Nordstrom, J. Sztipanovits, G. Karsai, and A. Ledeczi. Metamodeling - rapid design and evolution of domain-specific modeling environments. In *IEEE Conference and Workshop on Engineering of Computer Based Systems*, pages 181–187, March 1999.

[Ode95] James Odell. Meta-modeling. In *OOPSLA'95 Workshop on Metamodeling in OO*, October 1995.

[Ode96] James Odell. A primer to method engineering. In Brinkkemper et al. [BLW96], pages 1–7.

[OHSB01] A. L. Opdahl, B. Henderson-Sellers, and F. Barbier. Ontological analysis of whole–part relationships in OO–models. *Information and Software Technology*, 43:387–399, 2001.

[OMG01a] OMG. MOF specification version 1.3. formal/2001-11-02. Available at http://www.omg.org, November 2001.

[OMG01b] OMG. UML specification version 1.4. formal/01-09-67. Available at http://www.omg.org, September 2001.

[Ozk95] Esen Ozkarahan. Multimedia document retrieval. *Information Processing and Management*, 31(1):113–131, January–February 1995.

[PBDSB93] Phillipe Palanque, Remi Bastide, Louis Dourte, and Christope Sibertin-Blance. Design of user-driven interfaces using petri nets and objects. In C. Rolland, F. Bodart, and C. Cauret, editors, *Advanced Information Systems Engineering, CAiSE'93*, volume 685 of *Lecture Notes in Computer Science*, pages 569–585. Springer, 1993.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice-Hall, 1981.

[Pla92] Remi Planche. *Dominio de la modelización conceptual*. Masson, 1992.

[PM88] Joan Peckham and Fred Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):153–189, 1988.

[POB00] R. F. Paige, J. S. Ostroff, and P. J. Brooke. Principles of modeling language design. *Information and Software Technology*, 42:665–675, 2000.

[PW00] Jeffrey Parsons and Yair Wand. Emancipating instances from the tyranny of classes in information modeling. *ACM Transactions on Database Systems*, 25(2):228–268, June 2000.

[RAC99] G. Reggio, E. Astesiano, and C. Choppy. CASL-LTL: A CASL extension for dynamic reactive systems – summary. Technical Report DISI-TR-99-34, DISI–Università di Genova, Italy, 1999. ReggioEtAll99a.ps in ftp://ftp.disi.unige.it/person/ReggioG/.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object–oriented Modeling and Design*. Prentice Hall, 1991.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[RKR+00] Gianna Reggio, Alexander Knapp, Bernhard Rumpe, Bran Selic, and Roel Wieringa, editors. *Proceedings of the UML'2000 Workshop Dynamic Behaviour in UML Models: Semantic Questions*, 2000.

[RM83] N. Roussopoulos and L. Mark. A self–describing meta–schema for the RM/T data model. In *IEEE Workshop on Languages for Automation*, pages 147–153, 1983.

[Ros84] B.N. Rossiter. Data modeling and performance of data base systems. *Computer Physics Communications*, 33(1–3):13–21, 1984.

[RP91] J. F. Roddick and J. D. Patrick. Adding temporal support to the RM/T model. In *Proceedings of the 2nd Australian Databases-Information Systems Conference*, pages IX+427, 45–57, 1991.

[RSM95] Collette Rolland, Corine Souveyet, and Mario Moreno. An approach for defining ways–of–working. *Information Systems*, 20(4):337–359, 1995.

[Sae95] M. Saeki. Object–oriented meta modelling. In M. P. Papazoglou, editor, *Proceedings of the OOER'95, 14th International Object–Oriented and Entity–Relationship Modelling Conference*, volume 1021 of *Lecture Notes in Computer Science*, pages 250–259. Springer, 1995.

[Sal81] F. Saltor. A critique of RM/T. In *Prov. Convencio Informatica Llatina*, Barcelona, June 1981.

[Sal87] F. Saltor. Semantic relativism in databases: the case of RM/T. Technical Report 18/87, FIB, UPC, 1987.

[SC01] Keng Siau and Qing Cao. Unified Modeling Language (UML): A complexity analysis. *Journal of Database Management*, 12(1):26–34, January–March 2001.

[Sch01] Peter Scholz. Incremental design of statechart specifications. *Science of Computer Programming*, 40:119–145, 2001.

[SD98] Monique Snoek and Guido Dedene. Existence dependency: the key to semantic integrity between structural and behavioural aspects of objects types. *IEEE Transactions on Software Engineering*, 24(4):233–251, 1998.

[SGW94] Bran Selic, Garth Gullekson, and Paul T. Ward. *Real-Time Object-Oriented Modelling*. Wiley & Sons, 1994.

[SLTM91] K. Smolander, K. Lyytinen, V.-P. Tahvanainen, and P. Marttiin. Metaedit–a flexible graphical environment for methodology modelling. In R. Andersen, J. Bubenko jr., and A. Solvberg, editors, *Advanced Information Systems Engineering, CAISE'91*, volume 498 of *Lecture Notes in Computer Science*, pages 168–193. Springer-Verlag, 1991.

[SO94] X. Song and L. J. Osterweil. Experience with an approach to comparing software design methodologies. *IEEE Transactions on Software Engineering*, 20:364–384, 1994.

[Sow84] F. Sowa. *Conceptual structures, Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[SS77] John M. Smith and Diane C.P. Smith. Database abstractions: aggregation and generalization. *ACM Transactions on Database Systems*, 2(2):106–133, June 1977.

[Sta85] R. A. Stachowitz. A formal framework for describing and classifying semantic data models. *Information Systems*, 10(1):77–96, 1985.

[STW84] M. Schrefl, A. M. Tjoa, and R. R. Wagner. Comparison criteria for semantic data model. In *International Conference on Data Engineering*, pages 120–125, 1984.

[TCGB90] Luiz Tucherman, Marco A. Casanova, Pedro M. Gualandi, and Anelise P. Braga. A proposal for formalizing and extending the generalization and subset abstractions in the entity–relationship model. In F.H. Lochovsky, editor, *Entity-Relationship Approach to Database Design and Querying*, pages 27–40. Elsevier Science Publishers, 1990.

[tHV97] Arthur H. M. ter Hofstede and T. F. Verhoef. On the feasibility of situational method engineering. *Information Systems*, 22(6/7):401–422, 1997.

[tHvdW93] Arthur H. M. ter Hofstede and Th. P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10:65–100, 1993.

[TL93] J. P. Tolvanen and K. Lyytinen. Flexible method adaptation in CASE. The metamodeling approach. *Scandinavian Journal of Information Systems*, 5:51–77, 1993.

[Urb91] S.D. Urban. A semantic framework for heterogeneous database environments. In *First International Wokshop on Interoperability in Multidatabase Systems*, pages 156–163, 1991.

[UW02] Jeffrey D. Ullman and Jennifer Widom. *A first course in database systems*. Prentice Hall, 2002.

[vdB94] M. von der Beek. A comparison of statechart variants. In L. de Roever and J. Vytopil, editors, *Formal techniques in Real–Time and Fault Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 128–148. Springer, 1994.

[Ver93] T. F. Verhoef. *Effective Information Modelling Support.* PhD thesis, Delft University of Technology, Delft, The Netherlands, 1993.

[VtH95] T. F. Verhoef and A. H. M. ter Hofstede. Feasibility of flexible information modelling support. In J. Iivari and K. Lyytinen, editors, *Advanced Information Systems Engineering, CAISE'95*, volume 932 of *Lecture Notes in Computer Science*, pages 168–185. Springer, 1995.

[Wan96] Yair Wand. Ontology as a foundation for meta-modelling and method engineering. *Information and Software Technology*, 38:28–287, 1996.

[Wee91] R. A. Weedon. An object oriented extension to RM/T. Technical Report 91/14, Computing Department, Faculty of Mathematics, Open University, 1991.

[Wij91] G. M. Wijers. *Modelling Support in Information Systems Development.* PhD thesis, Delft University of Technology, 1991.

[Win87] A. Winter. Algorithms for retrieving or updating data of a medical database by means of an RM/T–based unirelational interface. In *Proceedings of the Seventh International Congress of EFMI*, volume I, pages LXXIX+1644, 569–573, 1987.

[Wir74] N. Wirth. On the design of programming languages. In *IFIP World Congress.* North-Holland, 1974.

[WK99] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Precise Modeling with UML.* Addison-Wesley, 1999.

[WR10] A. Whitehead and B. Russell. *Principia Mathematica.* Cambridge University Press, 1910.

[WtHvO92] G. M. Wijers, A. H. M. ter Hofstede, and E. van Oosterom. Representation of information modelling knowledge. In K. Lyytinen and V. P. Tahvanaimen, editors, *Next Generation CASE tools.* IOS Press, 1992.