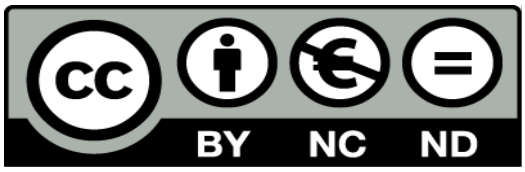
UNIVERSIDAD
DE LA RIOJA

TESIS DOCTORAL

Título
Modelización integral de la seguridad
Autor/es
Emilio Rodríguez Priego
Director/es
Francisco José García Izquierdo y Angel Luis Rubio García
Facultad
Facultad de Ciencias, Estudios Agroalimentarios e Informática
Titulación
Departamento
Matemáticas y Computación
Curso Académico



Modelización integral de la seguridad, tesis doctoral
de Emilio Rodríguez Priego, dirigida por Francisco José García Izquierdo y Angel Luis Rubio García (publicada por la Universidad de La Rioja), se difunde bajo una Licencia Creative Commons Reconocimiento-NoComercial-SinObraDerivada 3.0 Unported. Permisos que vayan más allá de lo cubierto por esta licencia pueden solicitarse a los titulares del copyright.

© El autor
© Universidad de La Rioja, Servicio de Publicaciones, 2014
publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

Universidad de La Rioja

Departamento de Matemáticas y Computación

Modelización integral de la seguridad

Tesis presentada por Emilio Rodríguez Priego
para obtener el grado de Doctor

Directores:
Francisco José García Izquierdo
Angel Luis Rubio García

2013

Agradecimientos

Cuando llegas al final esperado de una etapa es el momento de mirar atrás, reflexionar sobre como comenzó, la trayectoria que te ha llevado hasta aquí y hacer un balance de lo positivo o negativo de la experiencia. Recuerdo bien el instante en que decidí acercarme por la Universidad de La Rioja para comenzar esta aventura. En aquel momento todo era incertidumbre para mí. Dí el paso a pesar de que muchas personas me advirtieron que la elaboración de una tesis era algo duro y que me costaría mucho tiempo, esfuerzo y sacrificios. Cuando, a pesar de todo eso, decidí seguir adelante fue porque esperaba (más bien confiaba) en que terminaría siendo una experiencia interesante, enriquecedora y sobre todo satisfactoria.

Ahora puedo decir con mucha satisfacción que ha valido y sigue valiendo la pena y quiero dedicar unas palabras de agradecimiento a todos quienes lo han hecho posible.

La primera persona a quien quiero dar las gracias es a Julio. Sin lugar a dudas, de no haber sido por su apoyo, su confianza y su habilidad para ponerse en lugar de los demás, los inicios de mi doctorado habrían sido muy diferentes y probablemente no habría llegado hasta aquí. En segundo lugar quiero mostrar mi más sincera gratitud a Ángel Luis y Francisco. No exagero lo más mínimo si afirmo que ha sido para mí un lujo tenerlos de directores de tesis. Su trabajo ha sido impecable, no sólo han realizado una verdadera dirección de tesis, orientando con gran acierto mi trabajo, sino que además han demostrado una profesionalidad realmente excepcional como investigadores de la que he aprendido cada día. Les estaré siempre agradecido por su confianza, su apoyo y sobre todo por su constancia, su dedicación y su perfeccionismo que me ha empujado a intentar hacerlo lo mejor posible. Ellos han sabido cómo volver a levantar el vuelo en situaciones donde sin su tenacidad hubiera desistido a la primera.

Mi agradecimiento también quiero extenderlo a todo el Departamento de Ma-

temáticas y Computación. Sinceramente creo que es un modelo a seguir tanto dentro como fuera de nuestra Universidad. Además de la calidad profesional de quiénes lo forman, el ambiente que se respira en él es realmente envidiable y eso sin duda favorece la investigación y la docencia. Puedo decir que sin *vivir* aquí me he sentido como en casa.

Agradezco a Amelia su apoyo y su cariño. Su sacrificio y su comprensión han permitido que haya podido dedicar más tiempo a este trabajo. Por otro lado, su compañía en los viajes al extranjero para la presentación de ponencias fue una muestra de apoyo que tuvo gran valor para mí. También quiero dedicar unas palabras a mis hijos David y Andrés quienes han tenido que soportar mis ausencias sin entender muy bien por qué. Igualmente gracias a mi familia que, aunque desde la lejanía, también ha sabido darme ánimos para seguir adelante.

Finalmente me gustaría dedicar unas palabras a todos los amigos a quienes les gustaría honestamente leerse esta memoria más allá de esta sección. Si no lo hacen, no les guardaré rencor, me ha bastado con el interés y el apoyo que me han mostrado. Sé que la mayoría piensan que hay que estar un poco loco para realizar este esfuerzo, pero todos ellos saben que, como decía Machado, he disfrutado haciendo camino al andar, y lo he hecho con la satisfacción de que siempre he caminado acompañado por todas las personas a las que aprecio.
A todos, gracias de corazón.

Índice general

Agradecimientos **2**

1. Introducción **15**
1.1. Ámbito y objetivos generales 15
1.2. Trayectoria de la investigación 16
1.3. Estructura de la memoria 21

2. Modelo de seguridad integral SOA **25**
2.1. El concepto de seguridad integral en una arquitectura orientada a servicios 25
2.2. Aproximaciones a la seguridad en SOA 27
2.2.1. Modelos de seguridad 27
2.2.2. Modelos de SOA 31
2.2.3. Modelos de relaciones entre sistemas de información y seguridad 35
2.3. Nuestra propuesta de modelo conceptual de seguridad integral en entornos SOA 48
2.3.1. Modelo conceptual de SOA 48
2.3.2. Modelo conceptual de seguridad 53
2.3.3. Modelo conceptual de seguridad SOA 56
2.3.4. Conclusiones 64

3. Herramienta para el estudio de la modelización. La técnica RCM **67**
3.1. Definiciones 69
3.1.1. Definición (de *definición*) 69
3.1.2. Propiedades de las *buenas* definiciones 69
3.2. *Concept Maps* 70
3.2.1. Definición de *Concept Maps* 70

3.2.2. Características de los *Concept Maps* 71
3.2.3. Método de elaboración de los *Concept Maps* 72
3.3. References-enriched Concept Maps 74
3.3.1. Definición de RCM 74
3.3.2. Características de los RCM 74
3.3.3. Ejemplo básico de RCM 77
3.3.4. Autoría y roles en un RCM 79
3.3.5. Método de elaboración de RCM 80
3.3.6. Usos de los RCM 81
3.3.7. Generalización de los RCM a otros contextos 82
3.3.8. Métricas para el análisis basado en RCM 83
3.4. Técnicas de representación del conocimiento 88
3.4.1. Relación de las técnicas analizadas 89
3.4.2. Comparación entre *Concept maps* y RCM 91
3.4.3. Herramientas de diseño gráfico 92
3.4.4. Tablas comparativas 93

4. Principales dificultades en el estudio de la modelización 97
4.1. Dificultades en la comprensión de los conceptos básicos de la modelización 98
4.1.1. Modelo 98
4.1.2. Metamodelo 109
4.1.3. Lenguaje y Lenguaje de modelización 111
4.2. Dificultades sobre las relaciones entre modelización y orientación a objetos 113
4.2.1. El enfoque de OMG orientado a objetos sobre la modelización 114
4.2.2. Sobreutilización del concepto de Clase 116
4.2.3. Sobreutilización del concepto de Instancia 117
4.2.4. Carencia de diagramas orientados a objetos 119
4.2.5. Complejidad de modelo frente a sistema modelizado 120
4.3. Dificultades sobre los niveles de la modelización 122
4.3.1. Necesidad de un modelo raíz mínimo 126
4.3.2. Bidireccionalidad entre modelo y entidad modelizada . . . 126
4.3.3. Cardinalidad entre modelo y entidad modelizada 129
4.4. Dificultades sobre Metamodelización 130

4.4.1. Estereotipos UML 130
4.4.2. Herencia y metamodelización 133
4.5. Resumen 138

5. Principios sobre los fundamentos de la modelización 141
5.1. Principio de independencia. Triángulo de la modelización 144
5.2. Principio del enfoque científico. Definición y fases. 148
5.2.1. Formulación del problema y finalidad 149
5.2.2. Hipótesis 149
5.2.3. Predicción 152
5.2.4. Validación y experimentación 153
5.2.5. Comunicación 155

6. Fundamentos de la modelización 157
6.1. Protomodelos 158
6.1.1. Protomodelo 160
6.1.2. Entidades, objetos abstractos y conceptos 161
6.1.3. Oraciones, sentencias y proposiciones 163
6.1.4. Características de un protomodelo 166
6.1.5. Sistemas, temas (subjects) y originales 167
6.1.6. Triángulo de la modelización y protomodelos 167
6.2. El proceso analítico. Modelos analíticos. 169
6.2.1. Constructor analítico mediante ejemplo. Prototipos 174
6.3. Modelos sintéticos. El proceso sintético. 176
6.3.1. Constructor sintético mediante prototipos 180
6.4. Modelos completos 181
6.5. Modelos y analogía 184
6.5.1. El concepto de analogía. Modelos analógicos 184
6.5.2. Tipos de modelos analógicos 186
6.5.3. Analogía entre protomodelos y constructores. Interacciones. 188
6.6. Modelos 190
6.6.1. Definición general de modelo 190
6.6.2. Relación de modelización 190
6.6.3. Universo de modelos y originales 190
6.7. Sobre la naturaleza de lo físico y lo abstracto en la modelización . 191

6.7.1. Originales físicos, percibidos/concebidos/análogos y abstractos . . . 192
6.7.2. Modelos físicos y abstractos . . . 197
6.7.3. El problema de la identidad . . . 198
6.7.4. Modelos unitarios (singleton) . . . 199
6.8. Conclusiones y trabajo relacionado . . . 201

7. Relaciones entre modelos 205
7.1. Submodelos . . . 205
7.1.1. Comparación de submodelización y herencia . . . 206
7.1.2. Propiedades de submodelo . . . 207
7.1.3. Equivalencia entre modelos . . . 208
7.2. Metamodelos . . . 209
7.2.1. Elementos de metamodelo . . . 211
7.2.2. Propiedades de metamodelo . . . 217
7.2.3. Modelos universales. El modelo de sistema . . . 218
7.3. Modelos característicos de un modelo . . . 221
7.3.1. Propiedades de modelo característico de un modelo . . . 223
7.4. Transformación de modelos . . . 223

8. Modelos, lenguajes y lenguajes de modelización 227
8.1. Expresión de modelos . . . 228
8.2. El metamodelo lingüístico: modelos lingüísticos . . . 230
8.2.1. Definición de Metamodelo Lingüístico. Protomodelo . . . 231
8.2.2. Definición de Metamodelo Lingüístico. Constructores. . . . 233
8.2.3. Submodelos lingüísticos . . . 234
8.2.4. Características lingüísticas . . . 235
8.3. Correspondencia semántica . . . 236
8.3.1. Definición . . . 236
8.3.2. Nuestra propuesta de Triángulo de la modelización . . . 237
8.3.3. Tipos de correspondencia semántica . . . 238
8.3.4. Homomorfismo entre características semánticas y lingüísticas 239
8.4. Lenguaje . . . 239
8.4.1. Definicion . . . 239
8.4.2. Sublenguaje . . . 241
8.4.3. Lenguaje de modelización . . . 241

8.4.4. Metalenguaje 246
8.5. Resumen y conclusiones 248

9. Pragmática **251**
9.1. Roles de modelización 251
9.2. Modelizador independiente 253
9.3. Modelizador social 253
9.3.1. El proceso de la comunicación de modelos 254
9.4. Pragmática de los modelos software 260
9.5. Utilización de modelos 261
9.6. Existencia y destrucción de modelos 261

10. Revisión de los problemas de la modelización **263**
10.1. Dificultades en la comprensión de los conceptos básicos de la modelización 263
10.1.1. Modelo 264
10.1.2. Metamodelo 264
10.1.3. Lenguaje y Lenguaje de modelización 265
10.2. Dificultades sobre las relaciones entre modelización y orientación a objetos 266
10.2.1. El enfoque de OMG Orientado a objetos sobre la modelización 266
10.2.2. Sobreutilización del concepto de Clase y de Instancia . . . 266
10.2.3. Carencia de diagramas orientados a objetos 267
10.2.4. Complejidad de modelo frente a sistema modelizado 268
10.3. Dificultades sobre los niveles de la modelización 268
10.3.1. Necesidad de un modelo raíz mínimo 269
10.3.2. Bidireccionalidad entre modelo y entidad modelizada . . . 269
10.3.3. Cardinalidad entre modelo y entidad modelizada 270
10.4. Dificultades sobre Metamodelización 270
10.4.1. Estereotipos UML 270
10.4.2. Herencia y metamodelización 270

Conclusiones y líneas de trabajo futuro **273**

Bibliografía **279**

Índice de figuras

2.1. Escenario SOA Básico 26
2.2. Modelo de seguridad NSTISS (tomada de (McCumber, 1994)) . . 28
2.3. Modelo conceptual de referencia de SOA 32
2.4. Estándares SOA (tomada de (TheOpenGroup et al., 2009)) 33
2.5. Transformación de modelos en MDA (tomada de (OMG, 2003)) . 40
2.6. Relación entre MDA, MDD, MDE y MBE (tomada de (Brambilla et al., 2012)) 41
2.7. Enfoque WSbS*: compartición de estados y procesos en SOA . . . 49
2.8. Actor en SOA 52
2.9. PSC-Cert en SOA 59
2.10. PSO-Cert en SOA 62

3.1. *Concept map* representando las principales características de los *concept maps* (tomado de (Novak and Cañas, 2008)) 71
3.2. Ejemplo básico de RCM 78

4.1. Referencias del RCM de Modelo 99
4.2. Identificación y reformulación de definiciones en el RCM de Modelo (1 de 2) 100
4.3. Identificación y reformulación de definiciones en el RCM de Modelo (2 de 2) 101
4.4. RCM Model 102
4.5. Parámetros globales en RCM de Modelo 103
4.6. Lista de conceptos en RCM de Modelo 104
4.7. Lista de Grupos semánticos en RCM de Modelo 105
4.8. Definiciones del RCM de Modelo 106
4.9. Esencialidad de las definiciones en RCM de Modelo 107
4.10. Similitudes de definiciones en RCM de Modelo 108

4.11. RCM Metamodel 110
4.12. RCM Language 112
4.13. RCM Modeling Language 113
4.14. *InstanceSpecification* en MOF (tomada de (OMG, 2011a)) 119
4.15. *InstanceSpecification* en UML (tomada de (OMG, 2011b)) 119
4.16. Ejemplo de jerarquía de 4 niveles de OMG (tomada de (OMG, 2011b)) 123
4.17. Reutilización de *libraries* de UML en MOF (tomada de (OMG, 2011a)) 124

5.1. Triángulos del significado 146
5.2. Triángulos de la modelización 147
5.3. Versión inicial de nuestra propuesta de Triángulo de la modelización 148
5.4. Fases del método científico 151

6.1. Ejemplo modelo analítico 172
6.2. Ejemplo modelo sintético 178
6.3. Representación esquematica de un modelo completo 183
6.4. Tipos de modelos analógicos 187
6.5. Representación de los universos de originales y modelos 191
6.6. Ejemplo análisis físico y abstracto 194

7.1. Metamodelos 211

8.1. Reconocimiento facial de hombre o mujer o viceversa 228
8.2. Nuestra propuesta de Triángulo de la modelización 238
8.3. Representación gráfica de Lenguaje 240
8.4. Representación gráfica de Lenguaje de modelización 244
8.5. Representación gráfica de Metalenguaje 246

Índice de tablas

2.1. Dualidad Código/Estado entre actores 61
2.2. Resumen de certificados generados en el escenario de la figura 2.10 63

3.1. Correspondencia entre concept maps y RCM 77
3.2. Resumen de métricas sobre RCM 84
3.3. Resumen de técnicas de representación del conocimiento (1) . . . 93
3.4. Resumen de técnicas de representación del conocimiento (2) . . . 94
3.5. Resumen de técnicas de representación del conocimiento (3) . . . 95

4.1. Resumen de los problemas de la modelización 138

6.1. Correspondencia entre modelo humano y mecánico 197

9.1. Los roles de la modelización 252

Capítulo 1

Introducción

1.1. Ámbito y objetivos generales

La *modelización* es el medio que tenemos los seres humanos de comprender y controlar la complejidad del mundo que nos rodea. Aunque la mayoría de los seres vivos usan modelos, que han heredado de la evolución de su especie en forma de instinto, los humanos nos diferenciamos por nuestra capacidad para crear modelos complejos, gracias a los cuales el género humano ha llegado a ser superior al resto. Las personas hacemos uso de esta capacidad continuamente y comunicamos modelos a otras personas mediante un proceso de comunicación lingüística que también nos diferencia de otras especies. Además de este uso cotidiano, la modelización forma parte de cualquier disciplina científica. De hecho, la gran mayoría de los logros científicos han sido posibles gracias a la creación, uso y comunicación de modelos, en todos los ámbitos, especialmente en aquellos donde existe una mayor complejidad.

Uno de esos ámbitos es el de la seguridad. La *seguridad* es el grado de protección frente cualquier clase de peligro que pudiera producir un daño a un determinado sujeto. La complejidad asociada a la seguridad desde el punto de vista humano hace que sólo pueda abordarse mediante modelos, denominados *modelos de seguridad*. Existen infinidad de modelos de seguridad aplicables a diferentes campos.

La seguridad en las tecnologías de la información y las comunicaciones (TIC) es uno de los campos donde los modelos han sido investigados intensamente. Actualmente estas tecnologías juegan un papel muy importante en el modo en el que las personas se relacionan, trabajan, y en definitiva viven y por tanto su

seguridad condiciona inexorablemente la seguridad del ser humano actual.

Precisamente tras el estudio del caso particular de la seguridad en TIC, pudimos comprobar que existían (y aún existen) muchos aspectos de la modelización que requerían una investigación en profundidad. Esa fue la principal motivación y objetivo de esta tesis: el estudio de los fundamentos de la modelización.

Desde un primer momento, comprendimos que paradójicamente, mientras que la modelización ayuda a abordar la resolución de problemas complejos, la propia modelización, que es en sí misma un problema complejo, presenta muchas dificultades para ser abordada mediante un modelo sencillo de entender. Dicho de otro modo, si la modelización se utiliza para resolver problemas complejos, entonces por qué no abordarla mediante modelos. Precisamente, esta aplicación recursiva de un concepto en sí mismo (meta) es uno de los aspectos que más ha sido investigado en el contexto de la modelización, cuya comprensión ha dado lugar a eternos debates en la literatura, especialmente en lo que se refiere a sus relaciones con otros conceptos como la herencia o el lenguaje.

Sería pretencioso definir el objetivo de esta tesis como la creación de una nueva teoría de la modelización, puesto que un amplio conjunto de investigadores ha empleado largos años en el estudio de la teoría de la modelización, especialmente en el ámbito de las TIC, sin que hasta el momento exista un amplio consenso sobre la misma. Nuestro objetivo general por tanto, no es enunciar una nueva teoría sino mostrar algunas de las dificultades de las teorías de modelización existentes, tomando como punto de partida nuestra experiencia en la aplicación de modelos conceptuales de seguridad, y a partir de ahí proponer un enfoque diferente, que en nuestra opinión ayuda a entender mejor qué es la modelización y cuáles son las causas de algunas de las dificultades para su comprensión y aplicación.

1.2. Trayectoria de la investigación

La definición del objetivo mencionado no se realizó de forma directa. Las dificultades encontradas para avanzar en la definición de modelos conceptuales de seguridad fueron las que motivaron nuestra necesidad de profundizar en los fundamentos de la modelización.

En esta sección, introduciremos de forma general cuál ha sido la trayectoria de nuestra investigación con el fin de ofrecer una primera panorámica del proceso y la complejidad asociada a la misma investigación. De hecho, una de las decisiones

que tuvimos que adoptar para abordar esa complejidad fue, como indicaremos a continuación, la de desarrollar una herramienta de ayuda para facilitar el propio proceso de investigación en lo referente a la bibliografía.

Esta herramienta es una de las aportaciones de la tesis.

En un primer momento nuestra investigación estuvo centrada en los modelos conceptuales de seguridad en las arquitecturas orientadas a servicios, SOA (*Service Oriented Architecture*) tal como habían sido estudiados por la literatura, pero prestando mayor atención a la evaluación de los modelos según su carácter integral, es decir, su capacidad para dar respuesta a la seguridad de todos los elementos que pudieran ser vulnerables en un sistema SOA.

Se identificó que uno de los aspectos menos estudiados es la seguridad del código en SOA, es decir, cuando el código es transmitido por el consumidor o por el proveedor del servicio dando lugar a escenarios de código móvil. Como resultado de ese primer estudio se observó que era posible abordar la seguridad del código en SOA desde un punto de vista general con código móvil o no. Todo este análisis estuvo apoyado en el Modelo de referencia de las Arquitecturas Orientadas a Servicios (MacKenzie et al., 2006) publicado por el consorcio internacional sobre estándares de comercio electrónico y servicios web (OASIS) y que es el principal marco conceptual de referencia en SOA. En este contexto el concepto de modelo de referencia aparece como una herramienta útil para el estudio y la compresión de SOA y sirvió como punto de partida para el análisis de sus implicaciones sobre seguridad. No obstante, el término *modelo de referencia* tiene un significado más cercano al terreno terminológico (un lenguaje común) que al de la modelización, tal como indica (Oasis, 2011):

> The OASIS Reference Model for SOA provides a common language for understanding the important features of SOA but does not address the issues involved in constructing, using or owning a SOA-based system.

Esta definición ya plantea algunos interrogantes. Sirva como ejemplo la cuestión sobre cuál es la naturaleza de la relación que existe entre modelo y lenguaje o cómo el lenguaje proporcionado por ese modelo puede permitir comprender las características del objeto de nuestro estudio, cuando afirma que ese "..modelo de referencia proporciona un lenguaje para la comprensión de características importantes en SOA". Con el planteamiento de estos interrogantes intuimos, ya desde un primer momento, que la comprensión de algunos de los conceptos asociados a la modelización era una necesidad para todos aquellos quienes están utilizando o

van a utilizar modelos como herramienta de trabajo.

Esta definición también ilustra lo que fue el ámbito de nuestro estudio en esta fase preliminar con el uso de modelos (conceptuales) de referencia: la comprensión de las características de unos determinados conceptos.

El resultado de esta primera fase de estudio fue la proposición de un *modelo de referencia* (Rodriguez-Priego and Garcia-Izquierdo, 2007) sobre la seguridad SOA (sección 2.3) que complementa el modelo de referencia propuesto por OASIS con la consideración del código como un elemento esencial para alcanzar una seguridad integral en este ámbito. Ese modelo de referencia, que será descrito en el apartado 2.3.3, se completó posteriormente con su aplicación a datos y objetos en SOA (Rodriguez-Priego and Garcia-Izquierdo, 2009).

A continuación nuestro principal objetivo fue el de investigar el problema de la aplicación práctica de esos modelos de referencia (para código, datos y código+datos, es decir objetos), apoyándonos en los estándares de seguridad sobre servicios web, los cuales están en continua evolución. Debe tenerse en cuenta que aunque los modelos conceptuales son una herramienta útil, como su propio nombre indica se trata de modelos que se definen en un plano conceptual pero que deben ser desarrollados para que puedan ser aplicados en la práctica. Llegados a este punto, y tomando como referencia la extensa literatura estudiada sobre modelos de seguridad se planteó que, puesto que los modelos de referencia de seguridad SOA propuestos (apartado 2.3.3) tenían muchas características en común, parecía razonable obtener un metamodelo común a éstos. Dicho metamodelo debería permitir a priori un enfoque de Ingeniería del software Dirigida por Modelos (MDE) (*Model Driven Engineering*) (Brambilla et al., 2012), que actualmente es objeto de intensa investigación. De esta manera volvimos a encontrarnos con la modelización, no ya desde el punto de vista de modelos conceptuales de referencia, sino desde el punto de vista de los modelos orientados a unas determinadas arquitecturas.

Comenzamos entonces la investigación sobre las técnicas de modelización habitualmente utilizadas lo que condujo directamente a los estándares UML (*Unified Modeling Language*)(OMG, 2009a,b) y MOF (*MetaObject Facility*) (OMG, 2006a) propuestos por el *Object Management Group* (OMG). Se trata de estándares ampliamente usados tanto por parte de la industria como en la investigación sobre modelización. La comprensión de estos estándares para su aplicación al ámbito de nuestra investigación fue bastante dificultosa. En primer lugar, la documentación

es muy extensa (los documentos sobre UML y MOF en sus versiones actuales suman 1064 páginas de información en muchos casos redundante y confusa). Por otro lado, la literatura sobre esta materia conduce a una infinidad de trabajos relativos a la terminología, significado de los términos más básicos, problemas de inconsistencia, complejidad de los estándares, etc., en los que se pone de manifiesto la falta de consenso en cuestiones tan básicas como la definición de modelo o cómo aplicar determinados mecanismos propuestos por OMG como los estereotipos, los cuales junto con MOF parecían ser los mecanismos idóneos para la definición de metamodelos.

El resultado final de este arduo trabajo, en el que se revisó a fondo la extensa literatura sobre el tema, fue la identificación de un conjunto de problemas asociados a la teoría de la modelización y muy especialmente los relativos a los estándares propuestos por OMG. Se concluyó que estos problemas, que serán descritos en detalle en el capítulo 4, suponían un serio obstáculo en la comprensión, aplicación e investigación de la modelización (Rodriguez-Priego et al., 2010).

Fue en este momento cuando nuestro interés se centró en el estudio de los fundamentos de la modelización. La investigación de su teoría nos pareció un campo de estudio muy interesante debido, sobre todo, a que con nuestra experiencia inicial sobre los modelos conceptuales de seguridad en SOA teníamos suficientes cuestiones abiertas que merecían ser estudiadas. Uno de los principales obstáculos que nos encontramos en este proceso fue la dificultad de análisis de las propias definiciones de los conceptos importantes (modelo, metamodelo, lenguaje de modelización, etc.) puesto que existían muchas definiciones con diferentes matices y en algunos casos contradictorias. Por ese motivo decidimos utilizar alguna herramienta que permitiera la visualización de los conceptos utilizados en las definiciones de cada una de las referencias relevantes. Tras un estudio de los principales enfoques de análisis y representación del conocimiento se llegó a la conclusión de que la técnica de *Concept Maps*, tal como fue propuesta por Novak (Novak and Cañas, 2008), era la más adecuada para conseguir este propósito. Aunque no lo suficiente. Por esa razón, decidimos complementar dicha técnica para permitir además la visualización de las referencias y así poder analizar la relevancia de los conceptos asociados a cada definición. A dicha variación de Concept Maps la denominamos RCM (*Reference-enriched Concept Map*) (Rodriguez-Priego et al., 2013). Gracias a los RCM pudimos obtener algunas conclusiones interesantes sobre cuáles eran los conceptos relevantes en modelización, cuáles eran sus re-

laciones y qué referencias bibliográficas los utilizaban. Como explicaremos en el capítulo 3, en nuestra opinión los RCM, al estar basados en los Concept Maps, son además una herramienta válida aplicable en cualquier ámbito de la ciencia y la investigación.

Ayudados por los RCM, y con un conocimiento más profundo de la literatura sobre modelización, constatamos que los fundamentos de la modelización eran el principal problema, puesto que continuaban siendo cuestionados y debatidos durante un período que, desafortunadamente, se extiende durante al menos 10 años. Sirvan como ejemplo los importantes trabajos de Thomas Kühne y otros autores (Atkinson and Kühne, 2001, 2002a, 2003; Atkinson et al., 2000; Kühne, 2005, 2006a, 2009, 2010) en el período 2000-2010, centrados especialmente en metamodelización, herencia, lenguaje de modelización e instanciación. Por tanto, nuestra investigación se centró en la búsqueda de un enfoque que ayudase a responder a las principales cuestiones que reiteradamente quedaban sin respuesta o para las que la literatura no tenía una única respuesta. Comienza así una etapa de la investigación centrada en los fundamentos de la modelización motivada por la búsqueda de respuestas a preguntas tales como: ¿qué es un modelo? ¿Cuál es el papel de la orientación a objetos en la modelización? ¿Existe un modelo raíz? ¿Cuál es la diferencia esencial entre la metamodelización y la herencia? ¿Qué problemas plantea la elección del mecanismo de los estereotipos respecto al uso de un metametamodelo como MOF? ¿Cuál es la verdadera relación entre lenguaje y modelo? ¿Qué es un lenguaje de modelización y cuál es su relación con modelo y metamodelo? ¿Realmente *todo es un modelo* (Bézivin, 2005)?, etc.

Nuevamente se trataba de una ardua tarea, donde cada camino se desdoblaba en diversas ramificaciones y en donde a menudo los árboles no dejaban ver el bosque. Además se dan una serie de circunstancias que ejercían una negativa influencia a la hora de intentar abordar un nuevo enfoque hacia una visión más simple de la modelización:

- La mayoría de los autores con un empeño similar, y que podían servir como referencia, han terminado de una forma u otra centrando su trabajo alrededor del enfoque orientado a objetos propuesto por OMG, pero este enfoque de la modelización es bastante complejo de entender y aplicar. Entre las excepciones a esta mayoría de autores que siguen el enfoque OMG, podemos destacar la corriente de la Ingeniería del Método (Brinkkemper, 1996; Dominguez and Zapata, 2007).

- La adopción durante bastante tiempo del enfoque OMG ha producido cierta resistencia de la literatura a abordar visiones diferentes, así como a adoptar terminologías menos confusas (véase por ej. las afirmaciones de Kühne (Kühne, 2006a) sobre la utilización del término *instanciación*, procedente de la orientación a objetos en lugar de otros términos más cercanos a la modelización como *conforms to* defendidos por otros autores (Bézivin, 2005; Favre, 2004b).
- Se trata de una materia en la que confluyen diferentes disciplinas: computación, lingüística, semiótica, matemáticas, filosofía, psicología, epistemología, etc. Un ejemplo de propuesta en el que se muestran de manera explícita estas interrelaciones es la teoría de la *Fenomática* (Dominguez, 1999).

A pesar de estas dificultades y de que, como hemos indicado, nuestro objetivo no era desarrollar *otra teoría de la modelización*, el resultado de esta etapa es una propuesta para un nuevo enfoque de los fundamentos de la teoría de la modelización que resume y consolida gran parte de la investigación realizada hasta el momento. Su objetivo, como hemos indicado, ha sido el de clarificar las principales cuestiones planteadas, de tal manera que realmente pueda hablarse de un cambio de paradigma (Bézivin, 2005) pero desvinculado realmente del anterior paradigma de la orientación a objetos.

En línea con la literatura en la materia, para la redacción de esta propuesta (descrita en el capítulo 6 y siguientes) se utiliza esencialmente el lenguaje natural complementado con expresiones matemáticas básicas. Se pretende llegar a un compromiso entre lenguaje de apariencia formal y la claridad en la exposición, pues tal como se indica en (Harel and Rumpe, 2000):

> The degree of formality of a language is independent of its appearance.

La elaboración de esta propuesta inicial sobre las bases de la modelización nos permitió contar con una base conceptual sobre la que continuar nuestra investigación sobre otros aspectos tan importantes como lenguaje, sistema o pragmática.

1.3. Estructura de la memoria

La tesis está organizada en diez capítulos. A continuación resumimos el contenido de cada capítulo, con el fin de ofrecer una visión general de esta memoria, junto con una breve referencia a las principales aportaciones de esta tesis.

Capítulo 1 **Introducción**: el actual capítulo describe el ámbito y objetivos generales de la tesis, como se produjo la evolución de la investigación y la presente descripción de la estructura general de la memoria.

Capítulo 2 **Modelo de seguridad integral SOA**: describe los resultados de la fase preliminar de la investigación en la que se estudiaron diferentes modelos conceptuales relacionados con la seguridad en una arquitectura orientada a servicios y que sirvió para conocer los problemas de la modelización en un ámbito complejo y como motivación para el desarrollo de la investigación en modelización que es el ámbito de estudio de esta tesis. Las principales aportaciones en esta fase fueron las propuestas de modelos conceptuales de seguridad integral en SOA, tanto para código (Rodriguez-Priego and Garcia-Izquierdo, 2007), como para datos y objetos (Rodriguez-Priego and Garcia-Izquierdo, 2009). Estas propuestas fueron presentadas en los Congresos ICEWE 2007 y WebIST 2009 respectivamente.

Capítulo 3 **Herramienta para el estudio de la modelización. La técnica RCM**: como un primer paso para un conocimiento más profundo de los problemas terminológicos relacionados con las definiciones encontradas en la literatura sobre modelización, se desarrolla una nueva herramienta denominada RCM que es una de las aportaciones en esta fase, junto con un completo estudio comparativo de las principales herramientas de gestión del conocimiento (Rodriguez-Priego et al., 2013). Este trabajo ha sido publicado en la revista *Journal of Information Science* en 2013.

Capítulo 4 **Principales dificultades en el estudio de la modelización**: en este capítulo se describe como principal aportación, un análisis de los problemas de la modelización tras la revisión de un extenso conjunto de referencias y que fue expuesto y debatido en el Congreso MODELS 2010 (Rodriguez-Priego et al., 2010).

Capítulo 5 **Principios sobre los fundamentos de la modelización**: se trata de uno de los capítulos principales de la tesis que describe los principios sobre los que se sustenta el desarrollo de la teoría sobre los fundamentos de la modelización que se expone en los capítulos siguientes.

Capítulo 6 **Fundamentos de la modelización**: este capítulo constituye el núcleo de la tesis. Aborda las definiciones de los principales conceptos de modelización desarrolladas en base a los principios descritos en el capítulo anterior. Sus principal aportación es ofrecer un enfoque de la teoría de la modelización basándose en el método científico y que permite responder a las principales dificultades

de la modelización descritas en el capítulo 4.

Capítulo 7 **Relaciones entre modelos**: trata sobre las relaciones entre modelos abordándose cuestiones que han sido objeto de intenso debate durante años por la literatura (metamodelización, modelos raíz, herencia, etc.). La principal aportación de este capítulo es la de ofrecer igualmente respuestas a algunos de los problemas planteados en la literatura desde un nuevo punto de vista.

Capítulo 8 **Modelos, lenguajes y lenguajes de modelización**: este capítulo trata sobre lenguajes, pero no desde un punto de vista general sino enfocado desde la óptica de la teoría descrita en los dos capítulos anteriores. La principal aportación de este capítulo es nuevamente la de ofrecer respuestas a los problemas conceptuales relacionados con lenguajes, lenguajes de modelización, metalenguajes, etc.

Capítulo 9 **Pragmática**: en este capítulo se describen los mecanismos de comunicación de modelos para su aplicación en la práctica. Se estudian diferentes escenarios de aplicación apoyándose en los conceptos desarrollados en los tres capítulos anteriores. En este capítulo, la aportación más importante es un estudio exhaustivo de los mecanismos de comunicación de modelos.

Capítulo 10 **Revisión de los problemas de la modelización**: este capítulo retoma los problemas sobre la modelización planteados en capítulo 4 y describe resume las respuestas que ofrece nuestra teoría a estos problemas.

La tesis finaliza con un apartado de **Conclusiones y líneas de trabajo futuro**.

Capítulo 2

Modelo de seguridad integral SOA

En este capítulo vamos a describir el concepto de seguridad integral como hilo conductor de la primera parte de la investigación donde se analizaron las propuestas de modelos seguridad SOA en la literatura y que sirvieron como motivación hacia la definición de un enfoque general de modelización. En primer lugar definiremos el ámbito del problema y un conjunto de conceptos básicos sobre seguridad en SOA. A continuación repasaremos las principales soluciones encontradas en la literatura en este ámbito y finamente presentaremos nuestra propuesta de modelo conceptual de seguridad en SOA.

2.1. El concepto de seguridad integral en una arquitectura orientada a servicios

Uno de los principios fundamentales de la seguridad es su carácter integral. La seguridad que sólo centra su atención en un aspecto del recurso que quiere protegerse es una seguridad poco eficaz e incompleta. La seguridad de un sistema debe ocuparse de todos los elementos que lo constituyen puesto que de lo contrario la debilidad del sistema que quiere protegerse estará en aquellos aspectos menos protegidos o sin ningún tipo de protección. Resumidamente, este principio expresa que la seguridad general de un sistema viene determinada por la seguridad de sus elementos más débiles. Cada uno de esos elementos con un menor nivel de seguridad se denomina *vulnerabilidad* o más comúnmente *agujero de seguridad*.

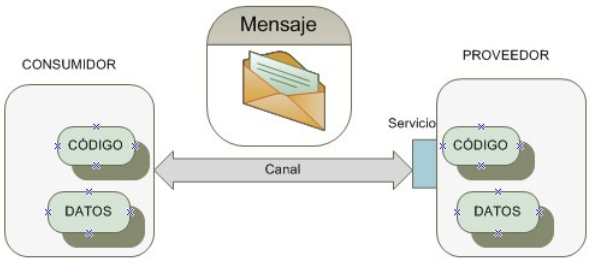


Figura 2.1: Escenario SOA Básico

Veamos el escenario más sencillo en una Arquitectura Orientada a Servicios representado en la figura 2.1. Se trata de un sistema en el que aparecen varios elementos que interactúan entre sí: el servicio, el consumidor del servicio, el proveedor del mismo, el canal, el mensaje y los datos y código mantenidos por el lado del consumidor y el proveedor. En un escenario más general, si los consumidores de servicios tuviesen que conocer directamente a los proveedores se necesitaría un número elevado de relaciones (una por cada par proveedor-consumidor). Para resolver este problema se introduce la figura del *intermediario (service broker)* de tal manera que por un lado el proveedor del servicio lo publica a través de dicho intermediario y por otro lado los consumidores encuentran a los proveedores igualmente a través de éste. Este nuevo elemento por sí solo introduce una mayor complejidad en el sistema pues el propio intermediario debe publicar servicios de búsqueda para que tanto proveedores como consumidores puedan relacionarse con él. El esquema básico *proveedor-consumidor-mensaje-canal-datos/código* se repite para cada par *proveedor-intermediario*, *consumidor-intermediario*, *proveedor-consumidor*. Sin embargo esta complejidad es inferior a la que se produciría si no existiese el intermediario y el número de proveedores y consumidores fuese elevado.

Para conseguir una seguridad integral en este nivel de abstracción cada uno de estos elementos debería ser securizado de alguna manera frente a las diversas amenazas a las que están expuestos para así evitar o disminuir sus vulnerabilidades. Los casos concretos pueden ser extraordinariamente complejos si se consideran las protecciones para cada una de las posibles amenazas y sobre cada uno de los elementos de un sistema SOA que pudieran ser vulnerables. Este problema ha sido tratado extensamente por la literatura, tal como se describe a continuación.

2.2. Aproximaciones a la seguridad en SOA

Una de las características de la modelización es obtener una visión simplificada de la realidad y, como hemos indicado, la seguridad es un concepto muy amplio y complejo, por tanto el problema de la seguridad y más concretamente en un escenario SOA es habitualmente abordado mediante modelos.

Un modelo de seguridad SOA está constituido intrínsecamente por tres modelos conceptuales: un modelo de seguridad (en el contexto de las TIC), un modelo de SOA y un modelo de la vinculación entre ambos. Existe una extensa literatura que trata de una u otra forma alguno o varios de estos modelos enfocados desde puntos de vista muy diversos. A continuación describiremos algunas de las propuestas más relevantes sobre estos modelos con el fin de tener una visión más precisa de la problemática asociada a la seguridad SOA.

2.2.1. Modelos de seguridad

Uno de los modelos clásicos de seguridad es el propuesto por el National Training Standard for Information Systems Security (NSTISS) (McCumber, 1994) estructurado en un modelo básico de tres dimensiones (tal como se muestra en la figura 2.2 tomada de (McCumber, 1994)) aplicable a cualquier sistema de información :

1) **Características de información críticas**: confidencialidad, integridad y disponibilidad (también conocido como CIA).

2) **Estados de información:** transmisión, almacenamiento y proceso.

3) **Medidas de seguridad:** tecnología, políticas y prácticas, educación, formación y concienciación.

El modelo propone un análisis de las *características de información críticas* 1) para los distintos *estados de información* 2) con el fin de identificar vulnerabilidades que puedan ser abordadas por *medidas de seguridad* 3) tanto técnicas como organizativas. La combinación de los tres niveles da lugar a 27 cubos que pueden ser analizados de manera diferenciada para obtener un balance de las vulnerabilidades de un sistema de información. Este modelo ha servido de referencia a diferentes autores para la definición de un modelo de seguridad SOA (véase por ejemplo (Jonnaganti, 2009; Sabbari and Alipour, 2011)) y cada uno de los niveles ha sido objeto de propuestas de mejora.

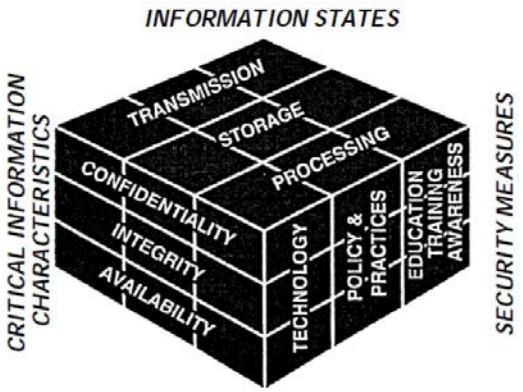


Figura 2.2: Modelo de seguridad NSTISS (tomada de (McCumber, 1994))

En el ámbito de la seguridad de la información, otra de las referencias más relevantes es la familia de estándares ISO/IEC 27000 (ISO, 2009) que adoptan el modelo CIA y centran su atención en la gestión de la seguridad en las empresas. Estos estándares están orientados a la certificación de los sistemas de gestión de la seguridad de la información (SGSI) y definen un proceso recursivo denominado PDCA (Plan-Do-Check-Act) cuyo objetivo es la mejora continua de la seguridad identificando los riesgos, las amenazas y las vulnerabilidades para así aplicar las medidas de seguridad adecuadas.

Todos los modelos de seguridad tratan de una forma u otra las dimensiones propuestas por NSTISS tal como describimos a continuación.

Características de seguridad

La primera dimensión ha recibido diferentes nombres (security principles (Fiere, 2007; Sabbari and Alipour, 2011), security objetives (Hafner and Breu, 2010), security concerns or domains (Johnston, 2004), aspects of security (Olovsson, 1992), security requirements (Jonnaganti, 2009; Opincaru, 2008; Parikh et al., 2006), dimensions of information security (ISO, 2009), security attributes (Savolainen et al., 2007), security goals (Menzel and Meinel, 2009), security concepts (Oasis, 2011)). A cada uno de los aspectos de la seguridad incluidos en esta dimensión los denominaremos *características de seguridad*.

Partiendo del análisis de Johnston (2004) veamos, descritas de una manera intuitiva, cuáles son las principales características de seguridad consideradas por la mayoría de los autores en sus modelos:

- **Identificación y autenticación** (responde a la pregunta *¿Quién es?*). En

un escenario de comunicación como SOA, la identificación de cada uno de los actores que intervienen es de vital importancia. Actualmente, una gran cantidad de amenazas a la seguridad provienen de las suplantaciones de identidad (por ejemplo phising), dificultades para realizar una autenticación de la identidad de una manera fiable (el usuario y la contraseña no son suficientes en muchos casos), etc. Debe observarse que la identificación de proveedor y consumidor del servicio no es suficiente. Este aspecto está relacionado con el concepto de zonas de confianza. Se trata de un enfoque que permite simplificar el proceso de autenticación evitando que cada uno de los intervinientes tenga que ser autenticado de manera continuada siempre que pertenezca a un determinado ámbito o zona de confianza.

- **Autorización** (*¿Qué puede hacer?*). Incluso en una zona de confianza donde los intervinientes *se conocen* es necesario establecer restricciones. El establecimiento de tales restricciones es a la vez un derecho y un deber de los intervinientes y su violación provoca en la práctica importantes problemas de seguridad. Las restricciones que tratan de las funciones que un interviniente puede realizar se denominan de *autorización*. Por ejemplo, en un servicio de banca electrónica el titular sólo podrá realizar determinadas operaciones como una transferencia o una consulta de un préstamo que le fue concedido, pero no podrá concederse un préstamo a sí mismo, puesto que esa operación sólo podrá realizarla una persona autorizada en la entidad bancaria.
- **Confidencialidad y privacidad** (*¿Qué puede ver?*). Otro tipo de restricciones, denominadas de confidencialidad, tiene que ver con qué recursos son visibles para un interviniente. Siguiendo el ejemplo anterior, los titulares de una cuenta bancaria podrán ver sus movimientos y éstos no serán visibles en general por otras personas (salvo por las personas autorizadas de la entidad bancaria). Cuando la confidencialidad está referida a la vida personal de un individuo, se denomina *privacidad*. De hecho, cuando el recurso es un dato de carácter personal su privacidad tiene enormes implicaciones legales y una enorme trascendencia en la práctica.
- **Integridad** (*¿Fue manipulado?*). Independientemente de si la información se mantiene privada y de si existen restricciones de autorización, en ciertas situaciones es necesario saber si esa información fue manipulada (intencionadamente o no). Por ejemplo, en la transmisión de mensajes, el receptor

debe tener la certeza de que la información recibida coincide con la enviada por el emisor. Otro ejemplo típico es la integridad asociada a una firma digital que implica que el documento firmado no ha sido alterado tras la firma (en ese caso la firma sería inválida).

- **Auditoría** (*¿Qué ocurrió?*). Este aspecto de la seguridad está relacionado con su dimensión temporal. Para que sea posible tanto la aplicación de medidas una vez ocurrido un hecho, como la investigación de posibles agujeros de seguridad a posteriori, debe guardarse algún tipo de registro de la actividad de los intervinientes para su análisis posterior. El registro en sí es a su vez uno de los recursos que deben protegerse para evitar su manipulación. Se trata de un aspecto de la seguridad muy importante e igualmente con gran impacto en escenarios reales.
- **No repudio** (*Yo no fui*). En este contexto, el repudio consiste en negar, por parte de alguno de los intervinientes, algún aspecto de un hecho en el que haya participado, tal como la identidad o el contenido del hecho en sí. Este aspecto de seguridad se apoya especialmente en las garantías de autenticación y de auditoría.
- **Disponibilidad** (*¿Estás ahí?*). Es la garantía de que los intervinientes pueden realizar sus funciones sin obstáculos. La principal amenaza de seguridad relacionada con este aspecto es la conocida como *denegación de servicio*. Su importancia radica en que en la práctica es uno de los ataques más utilizados por los mal denominados *hackers* (el término correcto es *crackers*). Existen infinidad de casos en los se han producido importantes pérdidas económicas provocadas por la denegación de servicio.

Esta lista no es exhaustiva. Otros autores han propuesto otras características de seguridad adicionales. En cualquier caso, el mayor nivel de seguridad lógicamente podrá alcanzarse analizando las vulnerabilidades del sistema respecto a cuáles de estas características sean relevantes conforme a unos objetivos de seguridad definidos.

Elementos securizables

La segunda dimensión hace referencia a los elementos del sistema que pueden ser vulnerables y por tanto susceptibles de ser securizados y también recibe diferentes nombres en la literatura (forms of security (Olovsson, 1992), security types (Fiere,

2007), informational tasks (Jonnaganti, 2009), domains (Sabbari and Alipour, 2011), security assets (Savolainen et al., 2007)). Nosotros emplearemos el término *elemento securizable* para referirnos al recurso que forma parte del sistema que se quiere proteger y al que están referidas las características de seguridad.

En el contexto de los sistemas de información y las comunicaciones, McCumber (1994) diferencia de un modo muy genérico tres tipos de elementos securizables: 1) transmisión, 2) almacenamiento y 3) proceso. Otros autores ((Fiere, 2007; Olovsson, 1992)) proponen diferenciar entre elementos de 1) hardware, 2) información y 3) administración.

Independientemente de las diferentes clasificaciones que puedan realizarse, los elementos securizables no son entidades aisladas sino que forman parte de uno o varios sistemas. El conjunto de sistemas conforman el dominio que quiere ser securizado. Debe resaltarse, tal como indican diversos autores (por ejemplo Breu et al. (2008); Fiere (2007); Olovsson (1992)) que la seguridad no sólo está referida a elementos tecnológicos sino también humanos (especialmente los organizativos).

Medidas de seguridad

Las medidas de seguridad (también denominadas security controls (Breu et al., 2008), security mechanisms (Menzel and Meinel, 2009; Olovsson, 1992), security solutions (Savolainen et al., 2007)) son aquellos mecanismos empleados para alcanzar un determinado nivel de seguridad sobre los elementos securizables que presentan vulnerabilidades.

En general, las medidas de seguridad pueden clasificarse en función del momento en el que son aplicadas (Olovsson, 1992): 1) Prevención 2) Detección y 3) Recuperación.

Savolainen et al. (2007) realiza una clasificación diferente distinguiendo 1) Mecanismos de control: forman parte de la funcionalidad del sistema securizable, 2) Propiedades de control: utilización de medidas de seguridad estándar y 3) Tolerancia a fallos: recuperación automática ante algún daño.

2.2.2. Modelos de SOA

El modelo básico de la arquitectura orientada a servicios es el representado en la figura 2.1, sin embargo, como ya se ha indicado, un modelo conceptual más completo, denominado *Modelo conceptual de referencia de SOA* o *SOA-RM* fue

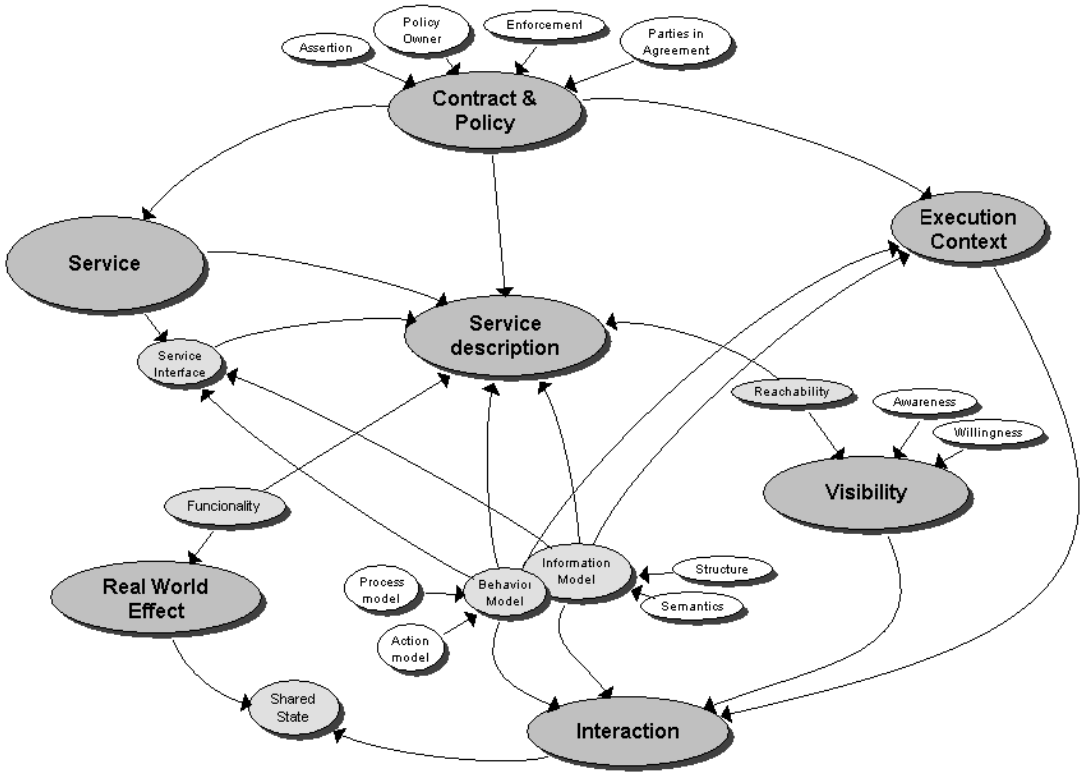


Figura 2.3: Modelo conceptual de referencia de SOA

publicado por OASIS (MacKenzie et al., 2006). Este modelo conceptual identifica los conceptos más relevantes de SOA y sus relaciones. A diferencia de otros modelos, SOA-RM no es expresado mediante el lenguaje UML sino que utiliza una versión simplificada de *concept maps*[1], tal como se aprecia en la figura 2.3. Como puede verse a continuación, SOA-RM ha servido de referencia a otros modelos de SOA propuestos por distintas organizaciones. Los esfuerzos de estandarización del modelo SOA han sido realizados principalmente por OASIS, OMG y The Open Group dando lugar a diferentes iniciativas, que aunque se diferencian en el enfoque, el nivel de abstracción y el grado de detalle, también tienen muchos puntos en común. En un intento de evitar la confusión, estas organizaciones publicaron un documento conjunto para clarificar los aspectos comunes y diferenciadores de cada estándard (TheOpenGroup et al., 2009). La figura 2.4, tomada de (TheOpenGroup et al., 2009), indica los principales estándares y los grupos de trabajo,

[1]Tal como indican los autores de SOA-RM, esta versión de concept maps no está basada en ninguna normativa y tiene como peculiaridad que no hay frases de enlace entre conceptos y que la flecha apunta al concepto (o conceptos) que depende de alguna forma del concepto origen. Como se verá más adelante, en esta memoria utilizaremos los concept maps con un enfoque y finalidad diferentes.

comités y grupos de interés asociados. A modo de resumen los estándares más

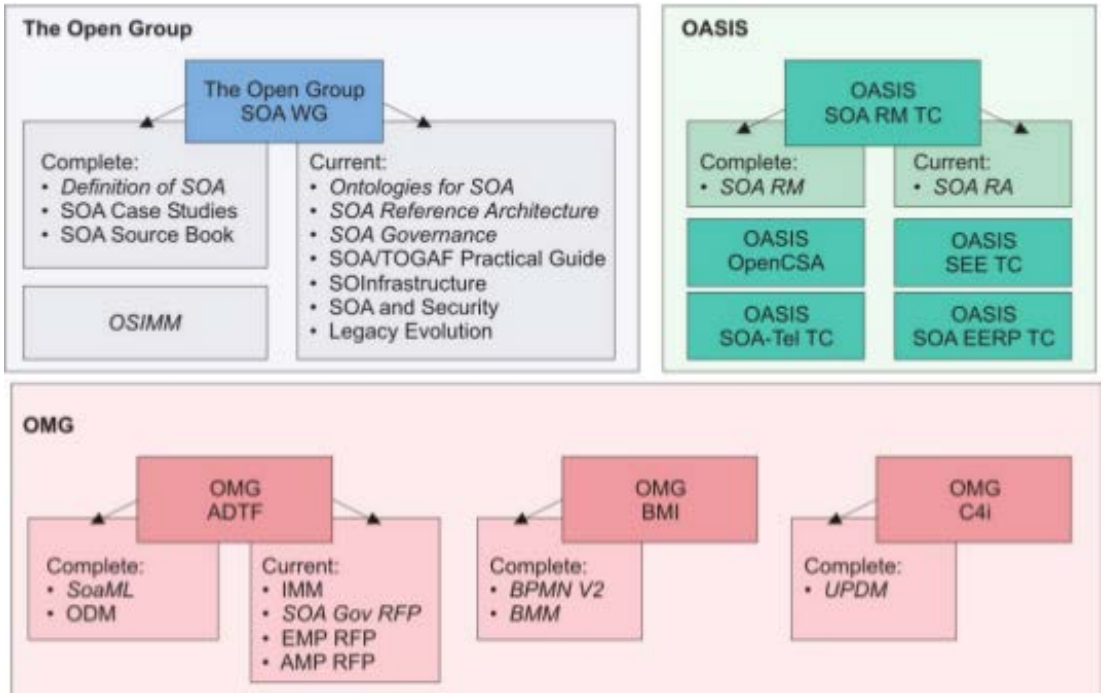


Figura 2.4: Estándares SOA (tomada de (TheOpenGroup et al., 2009))

relevantes son:

- **Reference Architecture Foundation for Service Oriented Architecture** (Oasis, 2011): se trata de una arquitectura de referencia basada en SOA-RM que permite comprender los elementos que interactúan en una arquitectura orientada a servicios desde diferentes puntos de vista y desde un nivel alto de abstracción.
- **SOA Reference Architecture** (The Open Group, 2011) (The Open Group): describe SOA en niveles. También permite comprender los elementos de SOA. Puede servir de referencia para el despliegue de SOA en la empresa.
- **Service oriented architecture Modeling Language - SOAML** (OMG, 2012b): proporciona un metamodelo y un *UML Profile* sobre la especificación y diseño de servicios en una arquitectura orientada a servicios.
- **Service Oriented Architecture Ontology** (The Open Group, 2010): al igual que SOA-RM permite comprender los principales conceptos de SOA de una manera más detallada.

Además de estos organismos, la literatura sobre la modelización de SOA es extensa (Amir and Zeid, 2004; Arsanjani, 2004; Benguria et al., 2007; Bruni et al.,

2009; Heckel et al., 2003; Jegadeesan and Balasubramaniam, 2008; Johnston, 2005; Lopez-Sanz and Marcos, 2012; López-Sanz et al., 2007; Swart et al., 2005). La mayoría de los autores utilizan UML como lenguaje de modelización y más concretamente la funcionalidad de los estereotipos (UML Profiles).

De todos ellos merece destacar el más reciente (Lopez-Sanz and Marcos, 2012) en el momento de escritura. Se trata de una propuesta de modelo SOA *alineado* con SOA-RM, expresado en un lenguaje específico o DSL (*Domain-Specific Language*) conforme a MOF y basado en la ingeniería dirigida por modelos (*Model-driven Engineering*), que describiremos más adelante. En opinión de los autores:

> Although in the beginning there was a notable trend towards extending UML as a way to define new DSLs, a close analysis of the evolution of initiatives in the field reported that some years later UML profiles were not taking off. As a matter of fact, many methodological proposals based on MDE were initially based on UML profiles. However, when researchers started to develop the technical support for their proposals, draw-backs such as inheritance of undesired features from UML or the inclusion of distracting elements in the models, became more apparent. As a result, those proposals, originally based on the use of UML profiles, moved towards the use of DSLs.

Como veremos a lo largo de esta memoria, nuestra investigación inicialmente condujo a unas conclusiones similares, las cuales sirvieron de motivación para proponer una respuesta no sólo a las deficiencias que actualmente presenta la modelización según el enfoque de OMG, sino a la modelización en general que aún carece de una teoría sólida que permita explicar tanto el enfoque OMG como el de los DSL.

A pesar de los problemas que plantea UML, Lopez-Sanz and Marcos finalmente optan por una solución mixta reconociendo que UML es un lenguaje de amplio uso e indicando que:

> (...) the plan is to first define the intrinsic characteristics of the DSLs for Software Architecture with services by defining a MOF-compliant metamodel. At a higher level of specification, after the metamodel has been refined, the UML notation is used for the concepts defined in that metamodel by defining concrete stereotypes and tagged values to be used in the different UML diagrams.

La propuesta, tal como reconocen los autores, mantiene muchas similitudes con el ya mencionado SOAML (OMG, 2012b), siendo la principal diferencia el grado

de abstracción (más alto en SOAML) y en su enfoque MDE.

En definitiva, tras la revisión de estas propuestas, las conclusiones más relevantes sobre este apartado son:

- Hay una gran variedad de modelos de SOA definidos en distintos niveles de abstracción (conceptual, arquitectura, etc.).
- La mayoría de los modelos utilizan el lenguaje UML como lenguaje de modelización y se basan en MOF, aunque crece la tendencia en la utilización de DSL sin perder la utilización de UML y MOF.
- A la vista de los diagramas publicados, la mayoría de los autores hace un uso básico de UML (diagramas de clases sin apenas atributos u operaciones) y algunos reconocen los problemas derivados del propio diseño de UML y su complejidad.

Si la literatura sobre seguridad y sobre SOA es abundante, no lo es menos la relativa a la seguridad de los sistemas de la información (de los cuales SOA es un caso particular), la cual se apoya de una manera u otra en los modelos de seguridad y del modelo de información en sí. A continuación haremos un repaso de la literatura sobre este tema en el que también se incluyen los trabajos de integración general de la seguridad en un sistema, que pueden ser aplicables cuando el sistema forma parte de una arquitectura orientada a servicios.

2.2.3. Modelos de relaciones entre sistemas de información y seguridad

Clasificaremos las investigaciones en dos grupos según si el enfoque es dirigido por modelos o no. Para cada uno de ellos y en la medida de lo posible resaltaremos cómo abordan los autores las dimensiones de la seguridad descritas en 2.2.1 y si se basan o no en alguno de los modelos de SOA indicados en la sección 2.2.2.

Modelos de seguridad de sistemas, no basados en MDE

En esta sección repasaremos los trabajos más relevantes sobre seguridad de sistemas, que no estén dirigidos por modelos. Una de las propuestas más relevantes es PWSSec (*Process for Web Services Security*) (Gutierrez et al., 2005) . Se trata de un proceso de integración de la seguridad de servicios web que permite a los desarrolladores incluir desde los primeros estadios del ciclo de vida software aspectos de seguridad en los sistemas de información basados en servicios web. Para

poder hacer esa integración los autores definen una arquitectura de referencia de seguridad basada en servicios web denominada WSSecArch. Centra su atención en la protección de mensajes SOAP, como elemento securizable, definiendo patrones sobre el estándar WS-Security (*Web Services Security*) de tal manera que las características de seguridad consideradas son autenticación, integridad y confidencialidad de mensajes.

UMLSec (Jürjens, 2002) es, junto con SecureUML (Lodderstedt et al., 2002) (que es descrito más adelante), una de las primeras propuestas sobre integración de seguridad en modelos, y más concretamente en modelos definidos mediante UML. Haciendo uso de UML Profiles, permite la anotación de modelos UML con características de seguridad tales como confidencialidad, integridad, no repudio, control de acceso y autenticidad.

Mediante un enfoque semiformal (Swart et al., 2005) plantea un modelo simple de SOA como punto de partida para evaluar, de una forma bastante original, la disponibilidad de los sistemas que permiten la ejecución de los servicios (desplegados mediante las denominadas unidades de despliegue).

Otra interesante propuesta es la de (Opincaru, 2008), la cual realiza un análisis de los diferentes enfoques para implementar la seguridad en SOA:

- **Seguridad integrada en la aplicación**: la implementación de la seguridad está codificada en la propia aplicación. Tiene como ventaja un mayor rendimiento pero carece de escalabilidad, reusabilidad y extensibilidad.
- **Seguridad integrada en el Middleware**: los aspectos de seguridad son manejados por la capa intermedia entre aplicación y servicios web. Su principal desventaja es que la implementación de seguridad no puede ser en general portada a diferentes middleware siendo por tanto complicada la conformidad de políticas de seguridad con servicios web distribuidos en distintos middleware.
- **Seguridad externa**: la seguridad es implementada mediante un sistema externo (por ejemplo un firewall XML). Tiene la ventaja de una mayor escalabilidad, reusabilidad y extensibilidad aunque a costa de un menor rendimiento (Tang et al., 2006).
- **Mixta**: combinaciones de los anteriores.

(Opincaru, 2008) concluye que el enfoque que ofrece mayores ventajas es el de la seguridad externa. Define una arquitectura de seguridad denominada SOSA

basada en este enfoque definiendo servicios externos de seguridad de autenticación, autorización, auditoría, criptografía, auditoría, entre otros, siendo el sistema securizable cualquier sistema SOA. Realiza una implementación Java basada en MuleESB [2].

Una solución similar es la planteada por (Baouab et al., 2009) aunque utilizando WS-BPEL (*Web Services Business Process Execution Language*) como base para la orquestación de los servicios de seguridad.

Las anteriores propuestas son de carácter general, pero puesto que la arquitectura orientada a servicios suele ser llevada a la práctica mediante la tecnología de los servicios web, la seguridad en la transmisión de mensajes es también uno de los aspectos más estudiados. O dicho de otro modo, el elemento securizable central es el mensaje.

Así, por ejemplo, (Mehr and Schreier, 2007) es una singular propuesta que aborda las características de confidencialidad e integridad de los mensajes SOA en la línea de UMLSec, pero además de OCL (*Object Constraint Language*), propone la utilización de un mecanismo poco conocido de UML denominado *UML Templates*.

(EL Yamany et al., 2010) es una solución en la misma línea que (Baouab et al., 2009; Opincaru, 2008) centrado en el mensaje como elemento securizable y en autenticación y autorización como las principales características de seguridad implementadas mediante servicios de seguridad.

(Bunge et al., 2008) centra su atención en el tráfico XML en red como elemento securizable respecto a diferentes características de seguridad (autenticación, auditoría, confidencialidad, etc.). Denomina FIX (Filtering to Inspect XML) a la técnica empleada en el nivel de aplicación de red.

(Rahaman et al., 2006) en la misma línea, analiza y ofrece una solución al problema del XML rewriting[3] que afecta al envío de mensajes en SOA.

Desde un punto de vista SOA, en la transmisión de mensajes no sólo se tiene en cuenta el canal de transmisión, que si está securizado protege lógicamente a los mensajes que circulan por él, sino que también se aborda la seguridad del mensaje si éste pasa por uno o más nodos intermedios. Los aspectos de seguridad

[2]MuleESB `http://www.mulesoft.org` es una de las plataformas software libre más utilizadas para la integración de servicios.

[3]XML rewriting es el nombre genérico con el que se conoce un tipo de ataques basados en la interceptación, manipulación y transmisión maliciosa de mensajes SOAP en una red de comunicaciones.

considerados suelen ser la confidencialidad, integridad, identidad y disponibilidad del mensaje a lo largo de todo su recorrido. Este ámbito de seguridad que va desde el emisor hasta el receptor se conoce como *seguridad de extremo a extremo*. Aunque han aparecido algunos estándares que formalizan la seguridad del mensaje de extremo a extremo (por ejemplo WS-Security), la adopción de estándares no es suficiente. Sirvan como ejemplo, el problema del rendimiento (Tang et al., 2006) o el tratamiento de ataques específicos de este ámbito como el ya mencionado XML rewriting (Rahaman et al., 2006).

Modelos de seguridad SOA basados en MDE

Un enfoque diferente es aquel en el que el modelo de seguridad, el del sistema de información y las relaciones entre ambos estén efectivamente separados (gestionados e implementados de manera independiente).

Algunos autores además proponen que la separación entre seguridad y recurso securizable sea en efecto consistente con la propia SOA: la seguridad es un servicio más (Baouab et al., 2009; EL Yamany et al., 2010; Opincaru, 2008). La idea es que la seguridad sea un aspecto externo al sistema que se quiere proteger y cuya aplicación pueda gestionarse según una determinada *política de seguridad*[4].

El enfoque dirigido por modelos (model-driven)

Aunque la vinculación entre los elementos del sistema de información y la seguridad podría hacerse de manera manual, la pregunta que se plantearon diversos autores fue si era posible realizar alguna conversión que permitiese *transformar* un sistema securizable en un sistema seguro de una manera automatizada mediante la declaración de los requisitos de seguridad establecidos en una política de seguridad. La publicación por OMG de la arquitectura dirigida por modelos *Model Driven Architecture (MDA)* (OMG, 2003) que dio lugar a la ya mencionada rama de la ingeniería del software MDE, ha inspirado muchos intentos de dar respuesta a esta cuestión y se ha consolidado como una de las vías más interesantes para abordar este problema. Su principal ventaja es precisamente que aplicada a la seguridad permitiría que no tuviese que estar embebida en el propio sistema sino que podría ser gestionada separadamente, aplicando el clásico principio de *separation of concerns*.

MDA es una propuesta muy ambiciosa, que incluso ha sido calificada de utopía

[4]Obsérvese que si la seguridad es un servicio entonces los propios servicios de seguridad también tienen que ser securizados mediante servicios (metaservicios de seguridad).

(Thomas, 2004). Básicamente propone un cambio radical en la forma de abordar el desarrollo del software similar al que se produjo con la introducción de los lenguajes de alto nivel para la generación de lenguaje máquina mediante un proceso de compilación. Se trata de dar un salto más arriba en el nivel de abstracción de tal manera que a partir de modelos se genere código (típicamente en un lenguaje de alto nivel, de tal manera que éste a su vez pueda ser compilado para su ejecución). El objetivo principal del uso de modelos en lugar de lenguajes de programación es abordar la creciente complejidad de los sistemas software basándose, como ya hemos mencionado, en la promesa de que un modelo ofrecerá una visión más simple del sistema modelizado.

El punto de partida es relativamente sencillo: se distinguen tres principales tipos de modelos (OMG, 2003):

- **Computation Independent Model (CIM)**: es un modelo de los requisitos del sistema y que algunas veces es llamado *modelo del dominio* o *modelo de negocio*. Este modelo es típicamente independiente de cómo esté implementado el sistema.
- **Platform Independent Model (PIM)**: modelo que describe el sistema, pero que no muestra detalles del uso de su plataforma.
- **Platform Specific Model (PSM)**: es un modelo del mismo sistema especificado por el PIM que especifica como ese sistema hace uso de la plataforma elegida.

Adicionalmente se define:

- **Platform model (PM)**: es un conjunto de conceptos técnicos, representando las diferentes clases de las partes que constituyen una plataforma y los servicios proporcionados por esa plataforma.

La figura 2.5, tomada de (OMG, 2003), ilustra el esquema básico de transformación tal como se define en la especificación. Prácticamente todo el documento se ocupa de describir las distintas posibilidades para la generación de un PSM a partir de un PIM. Una de ellas, que ha sido la preferida por la literatura, es el uso de metamodelos, que es descrito por MDA de la siguiente manera:

> A model is prepared using a platform independent language specified by a metamodel. A particular platform is chosen. A specification of a transformation for this platform is available or is prepared. This transformation specification is in terms of a mapping between metamodels. The mapping guides the transformation of the PIM to produce the PSM.

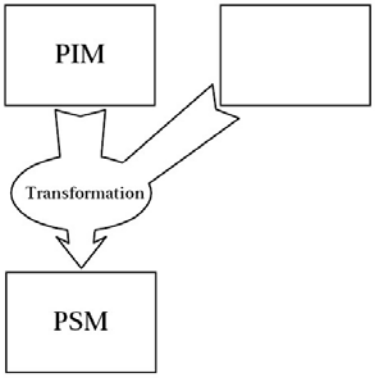


Figura 2.5: Transformación de modelos en MDA (tomada de (OMG, 2003))

La especificación de OMG suscita algunos interrogantes. Por ejemplo uno de ellos está relacionado con el CIM: si para obtener un PSM a partir del PIM se usa una transformación de modelos, ¿cuál es el camino para obtener un PIM a partir de un CIM? Esta cuestión ha sido abordada por algunos autores (por ejemplo, (Jeary et al., 2008; Karow et al., 2006)) quienes ponen de manifiesto cierta imprecisión en la definición de CIM y su relación con el proceso de construcción de software a partir de modelos. Otros pocos autores han intentado resolver este problema, véase por ejemplo (De Castro et al., 2011; Hahn et al., 2010) para SOA y sistemas multiagente respectivamente.

Otro aspecto que no queda claro es: ¿de qué naturaleza es esa *información adicional* (el cuadro blanco de la figura 2.5) de la que se alimenta una transformación para obtener un PSM a partir de un PIM?

Sin embargo, el mayor interrogante en la aplicación de la propuesta MDA es cómo definir los modelos en sí mismos mediante los lenguajes propuestos por OMG. La modelización, su significado, la definición de sus conceptos básicos, ha dado lugar a continuos debates que han ido sucediendo paralelamente a los intentos de aplicación de MDA por parte de las empresas. Una gran mayoría de estos debates ha puesto de manifiesto las confusiones derivadas de la complejidad y ambigüedad de los fundamentos de UML y MOF. Como ya hemos indicado, en la práctica quienes intentan modelizar siguiendo las directrices de OMG optan por usar las características menos complejas de UML o por definir un lenguaje específico (DSL).

Tras más de una década desde la propuesta, cabría esperar la existencia de un repositorio de modelos MDA reutilizables, es decir de CIM, PIM, PSM y, especialmente de PM, expresados en un lenguaje común (MOF o UML profiles según

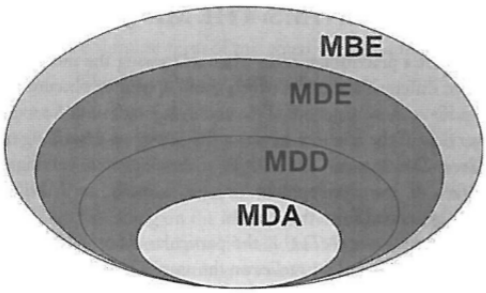


Figura 2.6: Relación entre MDA, MDD, MDE y MBE (tomada de (Brambilla et al., 2012))

las indicaciones de OMG). Existe una lista oficial de UML Profiles, publicada por OMG[5]. Sin embargo, en esa lista no encontramos los *Platform Models* de las plataformas más comunes: J2EE, .NET, PHP, Android, iOS, etc., los cuales serían los que cabría esperar para una aplicación práctica de MDA. Se echa en falta, por tanto, la existencia de unos PM ampliamente aceptados, estandarizados y reutilizables que sirvan como base para la aplicación de MDA y que permitan poner a prueba que realmente es posible generar diferentes PSM para esas plataformas. Todo ello a partir de un único PIM creado a su vez, a partir de los requisitos de un determinado CIM.

A pesar de lo anterior, MDA ha supuesto un antes y un después en el modo de enfocar la aplicación de la modelización a los problemas relacionados con la ingeniería del software. A partir de MDA, que fue inicialmente publicado en el año 2000, en 2003 se acuñó el término MDD para referirse al desarrollo dirigido por modelos y posteriormente en 2006 el término MDE para la ingeniería del software dirigida por modelos. Recientemente (Brambilla et al., 2012) propone también la utilización del término MBE (*Model-based Engineering*) para referirse a aquellos casos en los que los modelos software juegan un importante papel pero no son necesariamente los elementos clave en el desarrollo, es decir, no *dirigen* (*drive*) el proceso de desarrollo. En la figura 2.6, tomada de (Brambilla et al., 2012), se representa la relación entre estos términos. Efectivamente MDA inspiró y potenció la aplicación de la dirección por modelos en un ámbito más general (aunque MBE ha sido propuesto recientemente, MDE y MDD son superconjuntos de MDA y sin embargo fueron acuñados posteriormente, aunque ya estuviesen siendo usados antes de su definición).

[5]Puede accederse a esta lista desde http://www.omg.orgtechnology/documents/profile_catalog.htm

En adelante, puesto que MDE incluye MDD y MDA, lo emplearemos para referirnos a estos tres tipos de aplicación de la dirección por modelos. Cabría preguntarse si MDE ha tenido éxito en la práctica como sí puede afirmarse de la orientación a objetos. Como vemos a continuación no es una pregunta fácil de responder.

Aunque algunos autores reconocen cierta decepción respecto a la expectativas creadas a partir de MDA [6], hay diversos estudios de su aplicación en la práctica que indican cierto grado de éxito en determinados ámbitos.

(Hutchinson et al., 2011), realizado recientemente, presenta un exhaustivo trabajo de investigación basado en encuestas, entrevistas y visitas a empresas británicas que usan MDE. Sus principales conclusiones fueron las siguientes: 1) dentro de la muestra analizada la gran mayoría (83 %) tiene a priori una opinión positiva de MDE. Un 58 % piensa que MDE ha tenido éxito en la implantación. Por otro lado, un 5 % tiene una opinión negativa de MDE y un 20 % ha fracasado en su implantación; 2) el estudio es neutral respecto al tipo de dirección por modelos utilizada (MDA, MDD o MDE); 3) respecto a los lenguajes, el 85 % usan UML y el 40 % lenguajes específicos (DSL) de diseño propio. El 43 % piensa que UML es demasiado complejo y un 50 % cree que sirve para sus necesidades, existiendo una correlación entre quienes piensan que es muy complejo y los que creen que no es suficientemente potente; 4) respecto a las herramientas, existe una gran diversidad de ellas (unas 50 en la muestra analizada) que indica que aún deberá producirse una consolidación en ese mercado; 5) son interesantes también las conclusiones sobre la formación en MDE: un 47 % cree que MDE le permite contratar a ingenieros de software con menos experiencia, mientras que por otro lado el 74 % cree que MDE implica la necesidad de una formación extra; 6) respecto a los problemas asociados a la generación automática de código a partir de modelos, más de la mitad cree que es uno de los principales beneficios

[6] Por ejemplo J.Bezivin en http://modelseverywhere.wordpress.com/2011/10/17/why-did-mde-miss-the-boat/:

> Models have failed, at least temporarily. Model Driven Engineering (MDE) is a specific branch of software modeling that has been experimented for less than two decades. One of its best known instantiation is the MDA™ initiative launched in November 2000 by the OMG. The deployment of MDE seems today to have reached a standstill. In spite of the immense hopes that greeted the MDA initial proposal as a possible way to regenerate the entire software engineering practices, we must recognize today that its impact is rather limited and its perspectives quite confined (...)

de MDE, aunque tiene como contrapartida las dificultades para integrar el código generado en los proyectos ya desarrollados, así como para mantener sincronizados los modelos y el código.

Debe tenerse en cuenta que (Hutchinson et al., 2011) se centra sobre usuarios que *ya usan* MDE y por tanto no refleja la opinión de quiénes aún no conocen o usan MDE y cuáles son las razones por las que no se han acercado al mundo de la dirección por modelos. Por el contrario, (Torchiano et al., 2011) es un estudio similar centrado en la industria italiana y cuyo ámbito de estudio es todo tipo de empresa (no sólo las que ya han adoptado MDE). Aún así reconocen que una posible amenaza a la validez de su estudio es la autoexclusión de los desarrolladores no interesados en la modelización. El estudio recoge interesantes opiniones sobre diferentes cuestiones: 1) las grandes compañías y las micropymes usan en media la modelización siempre (un 20 %) y ocasionalmente (un 50 %), mientras que con las pymes ocurre prácticamente lo contrario; 2) las principales causas para no usar modelización son que requiere mucho esfuerzo y que no la encuentran suficientemente útil; 3) entre quiénes usan modelización un 44 % generan código y un 16 % producen modelos ejecutables; 4) un 76 % usan UML (de éstos un 11 % usan también UML Profiles) y sólo el 26 % usan DSL; entre ellos el 50 % usan notación textual, el 23 % gráfica y el 27 % mixta; 5) aproximadamente tres cuartas partes de gestores de proyectos, arquitectos de software y desarrolladores usan modelización, mientras que sólo el 11 % de expertos en un dominio determinado lo utilizan; 6) sólo el 10 % realizan transformaciones automáticas de modelos.

Un estudio anterior (Mohagheghi and Dehlen, 2008) realiza el análisis de los beneficios de MDE a partir de artículos publicados entre 2000 y 2007. Los autores concluyen que: 1) MDE está siendo aplicado a un amplio rango de dominios; 2) su estado está lejos de la madurez puesto que aún no existen herramientas integradas consolidadas y existen dificultades como la complejidad misma de la modelización, el desarrollo de PIMs portables y su uso dentro de sistemas heredados; 3) no existen evidencias cuantitativas del impacto de MDE sobre la productividad y la calidad del software salvo en algunos casos concretos (cita el caso de Motorola). Algunas afirmaciones de estos dos estudios parecen contradictorias, por ejemplo en (Hutchinson et al., 2011) se indica que parte del éxito de MDE está oculto, mientras que (Mohagheghi and Dehlen, 2008) afirma que existe más probabilidad de publicar los éxitos que los fracasos.

(Martinez et al., 2011) sigue una metodología parecida estudiando las pu-

blicaciones entre 2001 y 2010. Sus conclusiones más relevantes son: 1) aunque los estudios empíricos han aumentado en los últimos años el porcentaje de estudios que proporcionan evidencias empíricas sigue siendo comparativamente bajo; 2) los estudios en ámbito académico reportan aumento de productividad entre 2 y 20 veces superior respecto a otros paradigmas, mientras que en el ámbito industrial es más heterogéneo, yendo desde una pérdida de productividad del 10 % a una ganancia del 35 %; 3) el uso de herramientas especializadas reduce la curva de aprendizaje y aumenta la productividad; 4) existen evidencias de que MDD presenta deficiencias para expresar las reglas de diseño; 5) el uso de herramientas gráficas ayuda a la mantenibilidad de los sistemas, aunque hay estudios que indican justo lo contrario; 6) respecto a UML, aunque permite aumentar la reusabilidad lo hace a costa de una mayor complejidad y mantenibilidad; la complejidad es un problema para comprender y manejar un sistema software; dicha complejidad también se refiere a OCL; 7) sobre el impacto de MDD sobre la eficiencia, funcionalidad, fiabilidad, usabilidad y portabilidad, existen resultados poco contrastados y de difícil generalización.

Desde nuestro punto de vista pueden extraerse algunas conclusiones de dichos estudios:

- Aunque MDA fue el desencadenante del interés en la dirección por modelos, la mayoría de los estudios amplían el campo de estudio a MDE o a la modelización en general. Los estudios descritos no distinguen en el uso de MDA, MDD, o MDE por lo que es difícil evaluar el grado de éxito de la iniciativa.
- Los autores reconocen la dificultad de obtener datos objetivos y por tanto existe un grado más o menos relevante de subjetividad en los resultados sujetos a las interpretaciones de tanto las fuentes de la información, como de quiénes reciben dichos datos.
- Es necesario un estudio más global, no limitado a un país, y cuyo ámbito fuese también la mayoría de los potenciales usuarios de la modelización, no sólo los que ya la están usando.
- UML sigue siendo el lenguaje de modelización más utilizado, aunque su uso es predominantemente para sus funciones menos complejas. La complejidad de UML sigue siendo una barrera en la adopción de MDE. Aunque el uso de los lenguajes específicos (DSL) es considerablemente menor, su variedad indica que son utilizados para evitar el uso de las características más

complejas de UML.

- Hay una gran variedad de herramientas. Los autores concluyen que juegan un papel importante pero se trata de un mercado que aún tiene que alcanzar su madurez tendiéndose a la consolidación y predominancia de un conjunto más reducido de éstas.
- A pesar de su progresiva implantación en diversos ámbitos, todavía no se ha dado el paso de su uso generalizado, tal como ocurrió finalmente cuando se produjo la transición desde la programación estructurada a la orientada a objetos. [7].

Aplicación del enfoque de la dirección por modelos a la seguridad: Model Driven Security

Una vez descrito cómo aborda la literatura el tema de la dirección por modelos, veamos cómo se está aplicando este enfoque a la modelización de la seguridad de sistemas de información, del que la seguridad SOA es un caso particular.

(Breu et al., 2008) es una de las propuestas más relevantes sobre la aplicación de MDE a la seguridad. Aborda la seguridad en SOA desde un punto de vista MDE, según sus autores, *en el sentido de que integran estrechamente aspectos de seguridad con las vistas funcionales del sistema*. La solución se ha probado en la práctica mediante el proyecto health@net de ámbito sanitario. Propone un método de análisis de la seguridad que llama ProcSecO, en el que se distinguen varios niveles de abstracción. Este proceso plantea la identificación de dependencias, ingeniería de requisitos de seguridad, análisis de riesgos, definición de objetivos de seguridad, etc. Se basa en una plataforma dirigida por modelos para la configuración de servicios de seguridad, denominada SECTET (Hafner et al., 2006) que se centra en la seguridad de los procesos y que cuenta con algunas extensiones como SECTET-PL para definir políticas de seguridad (Alam et al., 2007). Usa UML Profiles, BPEL (Business Process Execution Language) y XACML (eXtensible Access Control Markup Language) permitiendo la generación de configuraciones de seguridad. Definen un modelo de seguridad relativamente simple con las características básicas de seguridad (integridad, confidencialidad y no repudio). Respecto al modelo SOA no se apoyan en ninguno de los estándares indicados en el apartado 2.2.2. Posteriormente SECTET fue mejorado por la plataforma

[7]Uno de los ejemplos más conocidos que demostraron la efectividad de la orientación a objetos es el de Tom Love que demostró cómo podían sustituirse 10.000 líneas de código en lenguaje estructurado por 202 líneas de código orientado a objetos (Love, 1993) (Ejemplo citado en http://www.infoq.com/news/2011/10/mde-missed-the-boat)

SECTISSIMO (Memon et al., 2008) que se apoya en el modelo SCA[8]. En una iniciativa similar (Saleem et al., 2010) propone conciliar las diferentes concepciones de seguridad por parte de expertos en procesos de negocio y expertos de seguridad, definiendo un lenguaje de modelización específico (DSL) que facilite al experto no técnico la definición de los objetivos de seguridad.

SecureUML (Lodderstedt et al., 2002) centra su atención en la generación de EJB DD (Enterprise Java Beans Deployment Descriptor) a partir de UML según un metamodelo definido con estereotipos. Respecto a su modelo de seguridad, aborda principalmente la característica de autorización, basándose en el modelo RBAC (Role-based Access Control). Los autores publicaron un artículo recopilatorio del desarrollo de esta propuesta a lo largo de una década (Basin et al., 2011), en el que ponen de manifiesto la necesidad de extender el enfoque más allá de la seguridad de las autorizaciones.

Además ha habido varios autores que han apoyado su solución en SecureUML: (Lodderstedt, 2004) desarrolla una solución completa de seguridad SOA mediante la aplicación SecureUML bajo el enfoque MDA. (Menzel and Meinel, 2010) propone un lenguaje denominado SecureSOA que permite definir la seguridad de un sistema SOA siguiendo el mismo planteamiento que SecureUML pero abordando otras características de seguridad (autenticación, no repudio, identidad, autenticidad, confidencialidad y confianza). Su aplicación de MDA consiste en la generación de código WS-SecurityPolicy. Una aplicación práctica de SecureUML puede encontrarse en (Clavel et al., 2008) donde se utiliza ComponentUML[9] como lenguaje para la modelización de sistemas basados en componentes. (Kaugers and Sukovskis, 2010) es un trabajo que se basa igualmente en SecureUML (y también en UMLSec) como lenguajes para la definición de Platform Independent Models siguiendo las directrices de MDA.

(Yskout et al., 2008) plantea una arquitectura en dos niveles (metamodelo y modelo) donde se define un modelo de requisitos de seguridad (conforme a un metamodelo de requisitos de seguridad) que combina con un modelo de arquitectura genérico (también conforme a su metamodelo) a través de un modelo de transformación (igualmente conforme a un metamodelo de transformación) para

[8]Service Component Architecture es una tecnología promovida por IBM y ORACLE y estandarizada por OASIS. Forma parte de una iniciativa más general http://www.oasis-opencsa.org para la simplificación del desarrollo de aplicaciones SOA con software de código abierto.

[9]ComponentUML es la denominación utilizada por (Clavel et al., 2008) para una versión simplificada del lenguaje de diagramas de clase UML.

generar un modelo de arquitectura seguro mediante el estándar QVT (Query/-View/Transformation) propuesto por OMG para la transformación de modelos. Desde el punto de vista de la seguridad, se trata de un enfoque sólo centrado en la característica de auditoría (los autores mencionan la posibilidad de extenderlo a otras características de seguridad).

(Wada and Suzuki, 2008; Wada et al., 2006) desarrolla una herramienta de transformación denominada Ark, basada en OpenArchitectureware[10], que toma como entrada un UML Profile, expresado en el lenguaje de intercambio *XML Interchange Language (XMI)* y obtiene como resultado código Java y descriptores de despliegue de la herramienta MuleESB. Se trata de una solución no centrada exclusivamente en la seguridad de SOA sino también aplicable a cualquier requisito no funcional de SOA. Comprende diferentes características de seguridad (confidencialidad, integridad, control de acceso). Se basa en un modelo simplificado de SOA propio.

IBM hace un planteamiento (Satoh et al., 2010) similar a SECTISSIMO en el que ponen el énfasis en el nivel de despliegue de las aplicaciones defendiendo, frente a otras propuestas, que la seguridad no se alcanza únicamente mediante la generación de código sino a través de la definición de descriptores de despliegue. Contemplan dos tipos de modelos de servicio: los basados en UML y en SCA mediante la cual se especifican los requisitos de seguridad. Las características de seguridad contempladas son autenticación, integridad, no repudio y confidencialidad. Para la transformación de los requisitos de seguridad del usuario en medidas de seguridad concretas se utilizan patrones (patterns).

Otra iniciativa procedente del ámbito empresarial es la de la empresa Object-Security que promueve el producto OpenPMF [11]. Se trata de una solución (Lang and Schreiner, 2008), que presenta como una mejora de XACML, está basada en MDA para la generación de políticas de seguridad partiendo de expresiones genéricas en un lenguaje comprensible.

Como hemos apuntado, la literatura es abundante y existen pocos trabajos de recopilación. Por ejemplo en (Kasal et al., 2011) se realiza un repaso donde faltan algunas referencias relevantes como SecureUML pero que pone de manifiesto

[10]OpenArchitectureware, www.openarchitectureware.org es un entorno de desarrollo dirigido por modelos basado en Java y que actualmente forma parte del Eclipse Modeling Project http://www.eclipse.org/modeling/

[11]Puede encontrarse más información sobre este proyecto en http://www.modeldrivensecurity.org

como la mayoría de las propuestas, aunque generalistas en su concepción inicial, se centran en un número limitado de características de seguridad, elementos securizables o medidas de seguridad.

2.3. Nuestra propuesta de modelo conceptual de seguridad integral en entornos SOA

En esta sección describiremos nuestra primera aproximación al modelo de seguridad integral en entornos SOA. Siguiendo el mismo esquema de la sección 2.2 en primer lugar se trata del modelo SOA, a continuación el modelo de seguridad y en tercer lugar el modelo de seguridad SOA.

2.3.1. Modelo conceptual de SOA

Nuestra aproximación (Rodriguez-Priego and Garcia-Izquierdo, 2007) a la definición de un modelo conceptual de SOA toma como base *SOA-RM* (MacKenzie et al., 2006). SOA-RM se basa en siete conceptos básicos: Servicio, Visibilidad, Descripción del Servicio, Contexto de ejecución, Efectos en el mundo real, Interacción y Contratos/Políticas cuyas relaciones pueden verse en la figura 2.3.

El concepto central del modelo es el *Servicio* que viene definido como el mecanismo que permite el acceso a un conjunto de una o más capacidades, en el que el acceso se proporciona por una interfaz normalizada y cuyo uso es ejercido consistentemente mediante restricciones y políticas especificadas en la definición del servicio.

Para nuestro estudio resaltamos tres aspectos de SOA-RM relacionados con la seguridad:

- **Opacidad del servicio**. En la descripción de Servicio, se establece que un servicio es típicamente opaco aunque no se prohíbe que la implementación sea visible para el consumidor.
- **Compartición del estado**. SOA-RM establece que como consecuencia de la invocación de un servicio se producen uno o más *efectos sobre el mundo real* que pueden ser de tres tipos: 1) información devuelta en respuesta a la petición del consumidor del servicio, 2) un cambio en el estado compartido de las entidades definidas y 3) alguna combinación de 1) y 2).

- **Contratos/Políticas del servicio**. Las políticas (requisitos establecidos por una de las partes) y los contratos (acuerdos entre dos o más participantes) que se aplican a Servicios pueden restringir los efectos del mundo real producidos al usar el servicio.

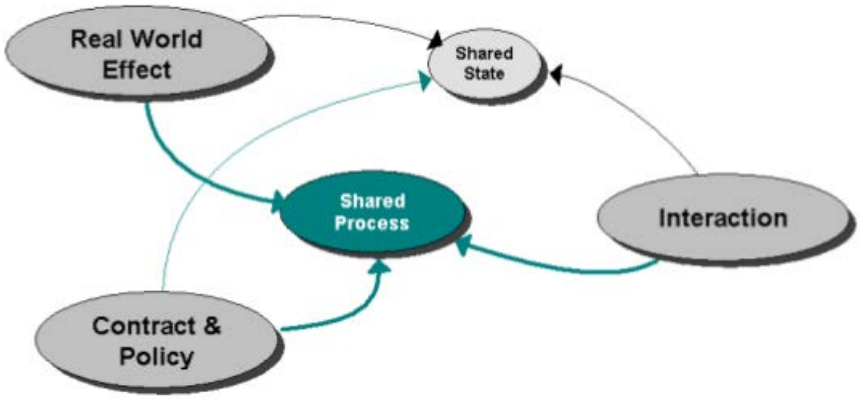


Figura 2.7: Enfoque WSbS*: compartición de estados y procesos en SOA

Teniendo en cuenta estos aspectos, nuestra propuesta fue introducir nuevos conceptos que amplíen SOA-RM para permitir abordar la seguridad del código en SOA:

- **El código como dato**: el código puede formar parte de la información intercambiada en el servicio, incluso, como veremos más adelante, puede tratarse del código que implementa el servicio.
- **Proceso compartido**: es el concepto análogo al estado compartido que indica SOA-RM, tal como puede verse en la figura 2.7 (en esta figura se indican en color verde los elementos que, en nuestra opinión, deberían ser añadidos a SOA-RM para que también sea tenido en cuenta el proceso compartido). De manera similar a cómo el proveedor y el consumidor comparten un estado que refleja un efecto en el mundo real, también puede definirse como efecto de un servicio la compartición del proceso entre el proveedor y el consumidor. Por otro lado, como también puede observarse en la figura 2.7 la interacción y los contratos y políticas, además de afectar al estado compartido, también afectan al proceso compartido.
- **Políticas y contratos sobre código**: las políticas y los contratos también pueden reflejar restricciones sobre el código como dato intercambiado (portabilidad) y/o como implementación de las acciones de un servicio.

El intercambio de código, la compartición del proceso y las políticas sobre el código aunque no son comunes en un entorno SOA típico basado en el uso de servicios web, puede producirse en algunos casos. Un ejemplo interesante es el de un proveedor que ofrece un servicio automático de monitorización de otros servidores. Para obtener un informe, el proveedor enviaría a dichos servidores un código que debe ejecutarse necesariamente en cada uno de ellos con el fin de monitorizarlos y éstos después de ejecutarlo devolverían el resultado al proveedor del servicio. Una vez obtenidos los resultados de cada uno de los servidores monitorizados, el proveedor obtendría un informe que sería la respuesta a la petición del consumidor del servicio. Como puede observarse, el proceso se reparte entre el proveedor del servicio y otros proveedores en los que éste se apoya para obtener el resultado, enviándoles el código necesario para conseguir resultados parciales que permiten a su vez obtener el resultado final para entregárselo al consumidor del servicio.

Por otra parte, el intercambio de código aparece habitualmente en la interacción entre usuarios y servicios de Internet, estando relacionado con los principales problemas de seguridad. Por ejemplo, en un escenario típico, un portal de Internet ofrece servicios a los usuarios accesibles a través de un navegador. La mayor parte del código se ejecuta en el lado servidor y por tanto está oculta al usuario. Sin embargo, hay otra parte del código que el servidor transmite al cliente (a través del navegador web) y que se ejecuta en la parte cliente (típicamente Javascript, Applets Java, Flash, etc.). En algunos casos, incluso se transmite el código completo del servicio para su ejecución off-line.

Aparentemente este escenario es diferente al que hemos descrito en un entorno SOA. Sin embargo en nuestra opinión la interacción con los servicios de Internet por parte de usuarios es sólo un caso particular de SOA en el que el lado cliente está controlado directamente por un usuario de manera interactiva en lugar de realizarse de manera automática. En efecto, la interactividad es lo que marca la diferencia puesto que los servicios web son típicamente no interactivos, aunque en última instancia sean ejecutados como consecuencia de la petición de un usuario (los sistemas que ofrece o usan servicios siempre lo hacen *en nombre de* un usuario).

Para reforzar esa idea, piénsese en el esquema de básico proveedor-intermediario-consumidor descrito en la sección 2.1. Podemos encontrar este esquema en multitud de ejemplos de interacción entre un usuario con los servicios ofrecidos por Internet. Las tiendas de aplicaciones para dispositivos móviles (Google Play, App

Store, etc.) son un ejemplo donde el intermediario ofrece un amplio catálogo de código realizado por diferentes proveedores que es transferido al consumidor para su descarga y ejecución. En este caso el *efecto sobre el mundo real* es del tipo 1) (información devuelta en respuesta a la petición del consumidor del servicio, por ejemplo la descarga de una aplicación en un dispositivo móvil), es decir, no se produce compartición de estado, ni tampoco de proceso.

Un ejemplo de compartición de proceso es el que se produce cuando un usuario accede a un portal a través de un navegador para la firma digital de documentos utilizando certificados digitales. El proceso completo se reparte entre el proveedor y el usuario (que actúa como consumidor), puesto que la firma del documento digital debe ser realizada mediante un código ejecutado en el consumidor (habitualmente un código Javascript o Applet Java ejecutado sobre un navegador web) para evitar que su certificado digital viaje a través de la red.

Estos ejemplos de compartición de código cuando se utilizan servicios de Internet de manera interactiva permiten entender mejor, mediante analogía, nuestra propuesta de tomar en consideración el intercambio de código y la compartición del proceso en un entorno SOA.

Veámoslo con un ejemplo. Supongamos un servicio de firma de contratos proporcionado por un determinado proveedor SOA. Los contratos son generados por el proveedor y tienen más de un firmante, pero la firma de los contratos no se hace de manera interactiva sino de forma automatizada mediante los respectivos certificados digitales de los firmantes. Estos certificados, como en el caso interactivo, sólo los poseen los firmantes, es decir no están en ningún caso en poder del proveedor del servicio. Cuando el consumidor del servicio solicita un contrato, el proveedor genera el documento y lo transmite secuencialmente a cada uno de los firmantes junto con el código que hace posible la firma. De este modo se evita que los certificados de los firmantes se envíen al proveedor del servicio, tal como ocurre en una firma interactiva en la que para que un usuario firme digitalmente un documento el proveedor genera el documento y le envía un applet que es el código que le permite firmarlo en local y enviarlo de vuelta al proveedor.

Esta analogía entre lo que se conoce como SOA y los servicios interactivos en Internet es la base de nuestro modelo conceptual de SOA, a partir del cual definiremos en el apartado 2.3.3 nuestro modelo de seguridad SOA.

Respecto a la implementación de SOA, incluyendo la compartición del código propuesta por nuestro modelo, la más común es la realizada mediante los conoci-

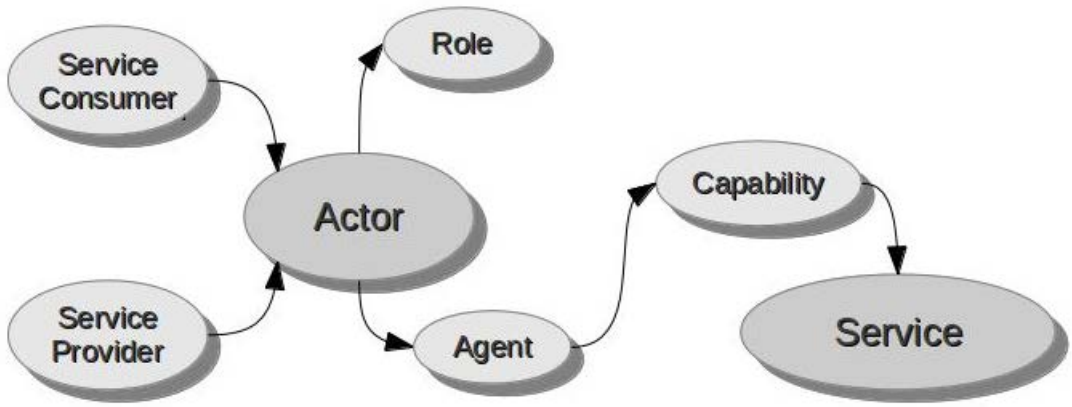


Figura 2.8: Actor en SOA

dos como servicios web. Existe un modelo de arquitectura de servicios web, Web Services Architecture (en adelante WSA) (Booth et al., 2004b) anterior a SOA-RM que no llegó a ser aprobado como estándar pero que define algunos conceptos (agente, actor, rol) que no aparecen definidos en SOA-RM y que son de interés para la definición de un modelo de seguridad SOA:

- **Agente**: programa que actúa en nombre de otra persona u organización (Booth et al., 2004a). Es sinónimo de Agente software (Booth et al., 2004a,b). Existen dos tipos de agentes según (Jacobs and Walsh, 2004):
  - **Agente de usuario**: pieza de software que actúa en nombre de una persona.
  - **Agente web**: persona o pieza de software que actúa sobre un espacio de información en nombre de una persona, entidad o proceso.
- **Actor**: Entidad física o conceptual (por ejemplo, personas, organizaciones, máquinas, software, etc.) que puede realizar acciones y que puede ser propietaria de agentes que solicitan el uso o proveen servicios web. (Booth et al., 2004a).
- **Rol**: Un actor puede asumir uno o más roles. Un actor en un nivel de abstracción puede ser visto como un rol en un nivel de abstracción inferior (Booth et al., 2004a).

Estos conceptos, expresados en la figura 2.8, ayudan a entender mejor cómo el modelo conceptual de SOA también puede ser aplicado al uso interactivo de servicios de Internet por parte de los usuarios.

Veamos un ejemplo: un usuario accede al portal www.loteriasyapuestas.es para

realizar una apuesta, el actor consumidor es el *usuario*, su rol es el de *apostante*, el agente (de usuario) es el *navegador web*, el servicio es *realizar apuesta* y el actor proveedor es la organización *Loterías del Estado*, siendo el sistema informático que da soporte a loteriasyapuestas.es su agente (web). Puesto que la apuesta es realizada mediante una miniaplicación (en flash) que se ejecuta en el ordenador del usuario, el proceso asociado al servicio (la apuesta) es compartido entre el agente del usuario y el agente del proveedor.

Supongamos que el usuario decide realizar una suscripción para una apuesta semanal automática aleatoria. En ese caso el actor consumidor sigue siendo el usuario pero cambia el agente, que es un software que genera la apuesta en su nombre y llama al servicio de realizar apuesta. En este caso, el proceso se realiza enteramente en el agente del proveedor.

Estos ejemplos muestran que SOA-RM puede verse desde un enfoque más general. En resumen, nuestra propuesta de modelo de SOA tiene las siguientes características:

- Extiende SOA-RM con los conceptos de compartición de proceso, actor, agente y rol.
- Permite modelizar las interacciones entre usuarios y servicios de Internet como un caso particular de interacción entre sistemas (típicamente mediante servicios web) al contemplar en ambos casos el intercambio y/o compartición de código y datos entre actores por la mediación de agentes.
- Facilita la definición de un modelo de seguridad SOA puesto que los conceptos de actor (así como los conceptos relacionados de rol y agente) tienen una gran importancia en el análisis de la seguridad y en la definición de modelos de seguridad ya que todas las propiedades de la seguridad descritas en la sección 2.2.1 (identificación, autenticación, autorización, privacidad, integridad, auditoría, no repudio y disponibilidad) están relacionadas directa o indirectamente con los conceptos de actor, rol y agente.

2.3.2. Modelo conceptual de seguridad

A diferencia del modelo de McCumber descrito en el apartado 2.2.1, nuestra propuesta de modelo conceptual de seguridad consta de cinco dimensiones:

1. **Elemento securizable**: (*qué*) elemento al que está referida la seguridad y que forma parte del sistema que se quiere securizar.

2. **Condiciones de seguridad**: (*cuánto*) cuanta seguridad se quiere alcanzar en un elemento securizable. Partiendo de las características de seguridad descritas en el apartado 2.2.1 y en base a un análisis de las amenazas y riesgos sobre cada elemento securizable se definen las condiciones de seguridad (por ejemplo, mediante políticas o contratos de seguridad).
3. **Medidas de seguridad**: (*cómo*) cómo se van a conseguir las condiciones de seguridad sobre cada elemento securizable.
4. **Actores de seguridad**: (*quién*) qué actores intervienen en ese elemento, qué papel (roles) desempeñan en su seguridad y qué compromisos y relaciones han establecido entre sí.
5. **Momento de aplicación**: (*cuándo*) a qué momento está referida la seguridad sobre el elemento securizable: antes (prevención), ahora (detección), después (recuperación).

Las tres primeras dimensiones están basadas en las del modelo NSTISS descrito en la sección 2.2.1 a las que se añaden dos nuevas dimensiones (actores y momento de aplicación) que en nuestra opinión son esenciales para abordar la seguridad de un sistema.

El modelo subsume lo tratado en la literatura (sección 2.2.1) exigiendo que se identifique *qué* quiere protegerse, *cuánta* seguridad quiere obtenerse, *cómo* conseguir esa seguridad, *quién* está involucrado en su obtención y *cuándo* es aplicable.

Los actores son una dimensión importante. Los actores (no mecánicos) representan el factor humano de la seguridad cuya importancia es cada vez mayor en el contexto de un mundo fuertemente interconectado a través de redes de comunicaciones como Internet. Aunque este factor aparece de manera implícita en la mayoría de los modelos de seguridad indicados en la sección 2.2.1, en nuestro modelo aparece explícitamente como dimensión pues de la misma manera que una condición de seguridad no tiene sentido si le falta la dimensión de a qué elemento está referida o cómo y cuándo es aplicable, tampoco tiene sentido sin los actores que participan. Por tanto, desde nuestro punto de vista sólo puede abordarse una seguridad integral si se identifican todos y cada uno de los actores (4) que pueden afectar a la seguridad del elemento securizable (1) y no sólo los actores finales[12]. Para ello un actor debe poder ser identificado y autenticado (para evitar suplantaciones) y su rol respecto al elemento securizable debe estar bien definido con

[12]Por ejemplo, los actores finales en SOA son el proveedor y el consumidor.

el fin de que puedan ser aplicadas las condiciones de seguridad (2) de autorización asociadas a ese rol. Los actores (generalmente a través de agentes) son los encargados de aplicar las medidas de seguridad (3) en los diferentes momentos de aplicación (prevención, detección y recuperación) (5).

Algunas consideraciones importantes al respecto son:

- Los actores forman parte del sistema y como tales son también elementos securizables, sobre los que se pueden definir unas condiciones de seguridad (identificación, autenticación) y unas medidas de seguridad (por ejemplo, uso de credenciales).
- La definición de actor como cualquier entidad que puede realizar una acción puede dar lugar a alguna confusión si no se tiene en cuenta que puede haber actores humanos y artificiales (por ejemplo, ordenadores). El concepto de agente, al estar identificado con agente software, ayuda a realizar la distinción más habitual entre personas (usuarios) y sistemas informáticos. Sin embargo, pueden existir personas que actúan en nombre de otras personas u organizaciones o agentes que actúan en nombre de otros agentes. Desde un punto de vista general, los actores tienen responsabilidades de las que se derivan acciones que pueden *delegar* en otros actores (personas o máquinas). Los agentes son un caso particular de *delegación*.
- La dimensión (2) pueden confundirse con las *características de seguridad* del apartado 2.2.1 . Desde nuestro punto de vista, las Condiciones de seguridad son un subconjunto de las características de seguridad aplicables a un elemento securizable determinado, obtenidas a partir de un análisis de los objetivos de seguridad que quieren aplicarse a ese elemento. Dicho de otro modo, una característica de seguridad se convierte en una condición de seguridad, según nuestro modelo, cuando un actor la ha definido como objetivo de seguridad respecto a un elemento securizable[13].
- Tal como se ha indicado, según (Booth et al., 2004a) un actor puede ser tratado como un rol en otro nivel de abstracción. Esta afirmación es consistente con la definición de actor como *entidad física o conceptual* aunque podría dar lugar a confusiones. Debe entenderse sin embargo que se trata de una cuestión lingüística puesto que usamos la misma palabra para nombrar

[13]Esta aclaración permite entender por qué en la literatura se utilizan diferentes nombres para los mismos conceptos (identidad, autenticación, integridad, etc.) en función del contexto en el que son tratados.

a una persona genérica y al rol que desempeña. En ese caso debe entenderse que las afirmaciones realizadas sobre ese actor son extensibles a todos los actores reales que asuman ese mismo rol. Por ejemplo, el actor *consumidor* representa a todas las personas que actúan con el rol *consumidor*.

- Como veremos a continuación, en la particularización del modelo a SOA, uno de los principales roles que puede asumir un actor es el de la *validación* de las condiciones de seguridad. Conforme a la quinta dimensión, esta validación puede hacerse en diferentes momentos: de antemano, en el momento en que puede producirse una amenaza o una vez que ya ha ocurrido un problema de seguridad.
- Un tipo especial de actores son las organizaciones[14] cuyas acciones siempre son delegadas en otros actores (personas o máquinas) y cuya estructura influye enormemente en la seguridad, tanto respecto a la toma de decisiones (gobernanza de seguridad) como la ejecución de las condiciones de seguridad (gestión de la seguridad) (Oasis, 2011).

2.3.3. Modelo conceptual de seguridad SOA

Una vez descritas nuestras propuestas de modelos conceptuales de seguridad y de SOA, en los subapartados siguientes trataremos los modelos conceptuales de seguridad SOA desde el punto de vista del código, los datos y la combinación de ambos (objetos). Finalizaremos este apartado verificando la conformidad de estos modelos respecto al modelo conceptual de seguridad propuesto.

Seguridad del código en SOA

Sobre la base del modelo conceptual de SOA ampliado con la compartición de los procesos y el modelo conceptual de seguridad descritos en las secciones anteriores, nuestra aproximación a la seguridad en SOA se centra inicialmente en el código como elemento securizable, tanto como dato intercambiado como aquél que da lugar a la ejecución de un servicio. Nuestro principal objetivo es definir un modelo conceptual a partir del cual sea posible construir arquitecturas concretas en las que la transmisión, recepción, carga, compilación, ejecución y validación

[14] En la legislación española, se usa incluso el término *persona* para referirse a las organizaciones (*personas jurídicas*) que son definidas como entidades capaces de contraer derechos y obligaciones y realizar actividades que generan plena responsabilidad jurídica, frente a sí mismas y frente a terceros.

del código sean servicios que puedan ser ofrecidos por sistemas potencialmente remotos y cuyas condiciones de seguridad sean establecidas/acordadas por los actores intervinientes mediante la creación de lazos de confianza entre ellos.

El modelo define los siguientes actores (que en este nivel conceptual se identifican con roles):

- **Autor**: creador del código y su propietario legal. Entre los servicios que puede ofrecer el autor se encuentran la mejora del código proporcionando nuevas versiones del mismo, la cesión del código a un tercero para su distribución, etc.
- **Suministrador**: proporciona el código a un consumidor o cliente y los distribuye con el permiso del autor.
- **Cliente**: usa el código proporcionado por un suministrador.
- **Validador**: verifica el código conforme a una política de seguridad previamente acordada entre el cliente y el suministrador, proporcionando pruebas de que se trata de un código seguro conforme a dicha política. Las comprobaciones que puede realizar el validador son muy diversas y no están limitadas por el modelo sino que son definidas conforme a técnicas específicas de validación, como por ejemplo, la integridad del código (Seshadri et al., 2005), su corrección respecto a reglas específicas tales como aserciones (Xu et al., 2000), inclusión de pruebas genéricas anexionadas al código (por ejemplo, Proof Carrying Code (Necula, 1997)), reglas de construcción de código seguro a nivel de código fuente (por ejemplo, WELL (Haldar et al., 2002)), reglas de verificación de código compilado (Necula and Lee, 1998; Xu et al., 2000), etc.
- **Procesador**: posee un entorno de ejecución en el que ejecuta el código proporcionado y devuelve el resultado de dicha ejecución. Un entorno de ejecución podrá ser virtual (por ejemplo, Java Virtual Machine, JVM) o físico (por ejemplo, una matriz de microprocesadores). Alguno de los servicios que el procesador puede ofrecer son: ejecución de pruebas de validación asociadas al código (Necula, 1997), servicios especiales tales como máquinas virtuales que facilitan la validación del código (Franz et al., 2003), etc.
- **Transformador**: a partir del código obtiene otro código mediante transformación funcionalmente equivalente y que es compatible con el entorno de ejecución proporcionado por el procesador. Esta transformación se conoce

habitualmente como compilación y por tanto utilizaremos indistintamente el término *transformador* o *compilador* cuando nos referimos a transformaciones de código. En general, entre los servicios ofrecidos por un *transformador*, se encuentran: optimización de código para un entorno de ejecución determinado, generación de código con características especiales (por ejemplo, preparado para su verificación por el validador (Necula and Lee, 1998)), etc.

Uno de los actores más relevantes es el *Validador*. En nuestra opinión, la seguridad de un sistema sólo puede ser garantizada a priori o a posteri mediante la comprobación de las políticas/acuerdos de seguridad establecidos.

Veamos un ejemplo respecto a la utilización de un servicio de Internet de manera interactiva: un usuario accede a un portal para la lectura de su correo mediante su agente de usuario (navegador web). Al tratarse de un acceso protegido mediante https, el navegador le presenta una pantalla para aceptación del certificado asociado al dominio al que está accediendo, por ejemplo mail.google.com, indicándole que el certificado asociado a ese dominio ha sido verificado por Thawte Consulting (Pty) Ltd. El usuario acepta dicha validación y procede a utilizar el servicio.

Este simple acto, cotidiano en el uso de Internet, presenta multitud de problemas de seguridad puesto que en la mayoría de los casos los usuarios aceptan dicha validación aún sin conocer (y por tanto confiar en) el actor que ha realizado la validación (es decir, aceptarían igualmente el acceso si el certificado estuviese validado por una entidad falsa). Se produce una cadena de confianza (google confía en Thawte, el usuario confía en Google y por tanto el usuario debería confiar en Thawte) que puede ser fácilmente rota en el momento en que uno de los eslabones de la cadena no haya adoptado las medidas de seguridad adecuadas.

Nuestra propuesta es que la validación esté referida a todos los actores que intervienen en la creación, generación y ejecución del código de un determinado servicio y que el resultado de dicha validación sea reflejado en un certificado (que denominaremos *Portable Secure Code Certificate* PSC-Cert) que pueda ser conocido por el consumidor del servicio como garantía de que las validaciones han sido realizadas conforme a las políticas/contratos del servicio.

Con el fin de que la ruptura de una cadena de confianza no comprometa la seguridad del proceso, las políticas/contratos de seguridad deben establecer si existe más de un circuito de validación (que se vería reflejado igualmente en el

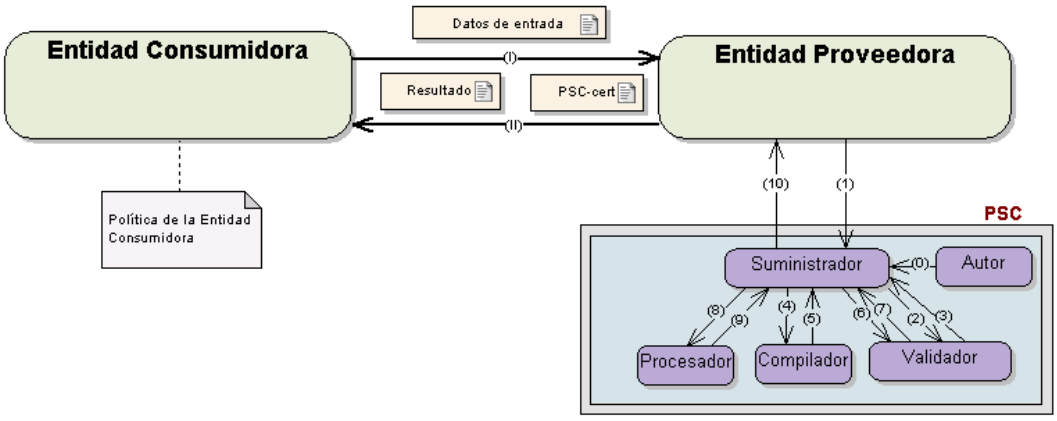


Figura 2.9: PSC-Cert en SOA

PSC-Cert).

Mediante este enfoque, el PSC-Cert refleja que un código es cualificado como seguro puesto que ha sido creado (autor), suministrado (suministrador), verificado (validador) y transformado (compilador) por entidades identificadas y de confianza. Así mismo, los metadatos incluidos en el PSC-Cert permiten su comprobación por un tercero fuera del círculo de confianza creado entre los actores.

El PSC-Cert es un contenedor de metadatos de seguridad que no tiene una estructura fija. La estructura, el formato y el contenido del PSC-Cert son acordados por las entidades proveedora y consumidora del servicio conforme a la política de seguridad acordada. Veamos un ejemplo de utilización de PSC-Cert en SOA, según el modelo conceptual propuesto.

Consideremos una entidad consumidora de un servicio ofrecido por una determinada entidad proveedora. Supongamos que es la entidad consumidora quien establece una política sobre las medidas de seguridad que deben ser soportadas por los proveedores de los servicios que desea usar, entendiéndose que para usar un servicio elegirá sólo aquellos proveedores del mismo que cumplen su política de seguridad.

Según la WSA, la expresión de esta política de seguridad se expresa generalmente mediante un conjunto de estándares tales como WS-Policy, WS-PolicyAttachment, WS-SecurityPolicy, WS-Security, etc. (Weerawarana et al., 2005). Nuestra propuesta es que se incluya el código como elemento securizable además de los ya habituales de canal de comunicaciones, datos y mensajes.

Este nuevo escenario, representado en la figura 2.9, sería el siguiente:

(I) la entidad consumidora, cuya política de seguridad establece que debe so-

portarse código seguro mediante PSC-Cert, solicita un servicio de la entidad proveedora. Antes de procesar el servicio, la entidad proveedora, si no lo ha hecho anteriormente, realizará la implementación del servicio siguiendo los siguientes pasos (0 a 9 en la figura) para el PSC-cert de dicha implementación: (0) El autor crea el código y lo envía a un suministrador para su distribución. (1) La entidad proveedora localiza y solicita el código al suministrador (2) El suministrador pide al validador la verificación del código conforme a la política establecida por la entidad proveedora. (3) El validador entrega el resultado de la validación. (4) Si el código suministrado no ha sido compilado para la arquitectura en la que va a ser procesado, el suministrador solicita al compilador su generación (y eventualmente, su optimización para esa arquitectura). (5) El *transformador* devuelve el código compilado. (6) Eventualmente, el suministrador puede solicitar al validador que compruebe si el código generado es correcto conforme a la política establecida. (7) El validador devuelve el resultado de la validación del código compilado. (8) El suministrador solicita al procesador la ejecución del código indicándole el entorno de ejecución de la entidad proveedora (9) El procesador devuelve el resultado del procesamiento del código al suministrador. (10) El suministrador entrega el resultado de la ejecución a la entidad proveedora. Se trata de un ejemplo de flujo genérico que se realiza con la frecuencia establecida en la política. Nótese que pueden darse excepciones si alguna de las validaciones no tiene éxito.

(II) la entidad proveedora devuelve a la entidad consumidora el resultado del proceso junto con el PSC-cert, que será comprobado por ésta examinando si cumple los requisitos de seguridad que su política establece. Así por ejemplo podrá verificar si cada uno de los actores son entidades de confianza, cuáles son los métodos de verificación empleados por el validador del proceso, etc. En este ejemplo, no se produce intercambio de código, pero en caso de que exista compartición del proceso, la entidad proveedora podría enviar el código del servicio (o parte de él) para su proceso en la entidad consumidora, junto con el PSC-Cert. Otra posible variación es la portabilidad del código pero con la premisa de que la entidad consumidora no conozca el código sino su nivel de seguridad. Para ello, por ejemplo, la entidad proveedora envía a la entidad consumidora una copia del código cifrada con la clave pública del validador que permite que éste descifre el

código para su validación sin que la entidad consumidora lo conozca. Pruebas periódicas de validez y el uso alternativo y simultáneo de diferentes validadores de confianza pueden incrementar aún más el nivel de seguridad exigido al código.

Es importante recalcar que el PSC-Cert refleja la política establecida por una de las partes (generalmente el consumidor) exigiendo un nivel de seguridad determinado (bastante elevado en el ejemplo). En la práctica existirán procesos más simplificados, pero con el único requisito de que para que pueda obtenerse un PSC-Cert debe existir un proceso de validación.

	Código	**Estado**
Autoría	Autor	Propietario
Suministro	Suministrador	Suministrador
Transformación	Compilador	Convertidor
Proceso	Procesador	Gestor de datos
Validación	Validador	Validador

Tabla 2.1: Dualidad Código/Estado entre actores

Seguridad de los datos en SOA

El modelo descrito puede extenderse no sólo al código sino a los datos. El mismo proceso descrito para el código puede aplicarse a los datos, distinguiendo actores análogos a los del código, tal como se indica en la tabla 2.1. Así por ejemplo, una conversión de datos será realizada por un Convertidor (análogo a un compilador) y el proceso de los datos es realizado por un gestor de datos (por ejemplo un gestor de bases de datos). Denominamos PSS-Cert (Portable Secure State Certificate) al certificado así obtenido.

Seguridad de objetos en SOA

El siguiente escenario es aquél donde son portables el conjunto de código y datos (lo que podríamos denominar objetos). En ese caso el certificado, que denominamos análogamente PSO-Cert (Portable Secure Object Certificate) obtenido como la agregación del PSS-cert del estado, los PSC-cert de sus métodos, y un tercer tipo de certificado que refleja la validación de que el estado sólo es accedido por sus métodos conforme al paradigma de la orientación a objetos, denominado PSMS-cert (Portable Secure Method-State Certificate) En el ejemplo de la figu-

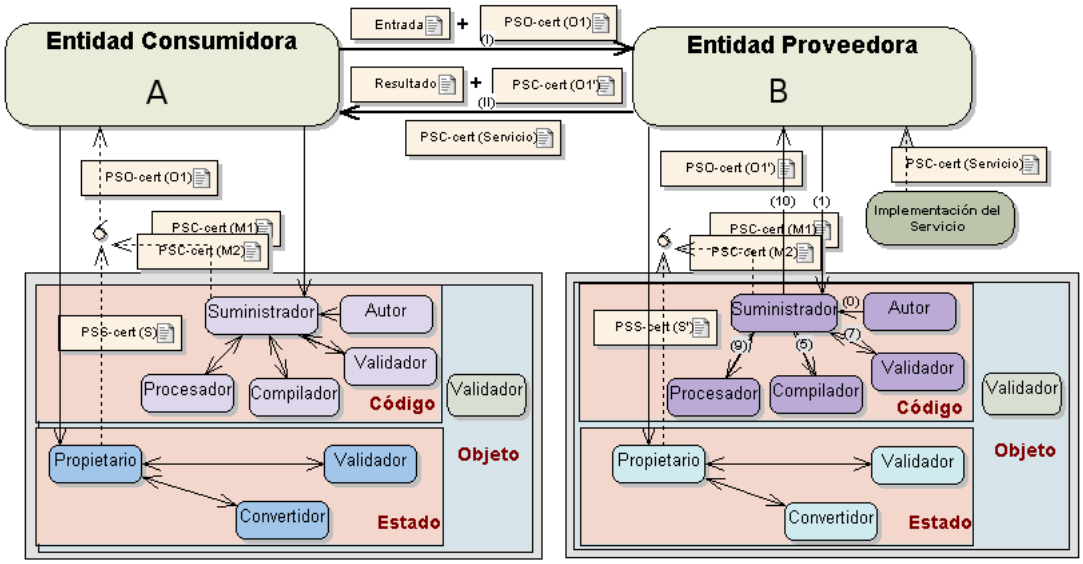


Figura 2.10: PSO-Cert en SOA

ra 2.10 se intercambian tres diferentes certificados en el uso de un servicio. La entidad consumidora A solicita un servicio de la entidad proveedora B enviando junto con su solicitud un objeto de entrada O_1 con un estado S_1 y dos métodos M_1 y M_2 conforme a la política establecida por B. A y B utilizarán PSO-certs si así está establecido en respectivas políticas/contratos y consecuentemente ambas entidades confían en los actores que intervienen en la obtención de los PSO-cert.

Una vez que B recibe $PSO(O_1)$ obtiene el estado, el código de las operaciones M_1 y M_2, y los certificados de estos elementos. Se entiende que el objetivo final es que tanto A como B puedan tener confianza entre sí: A porque no puede permitir que los métodos M_1 y M_2 sean ejecutados en un entorno que no sea de confianza (ya que en ese caso podría comprometerse el objeto en sí) y B porque tampoco puede permitir que se ejecuten en su entorno de ejecución operaciones que puedan suponer un riesgo al ser ejecutadas. Por esa razón, además del $PSO-cert(O_1)$ que A envía a B para garantizarle que procede de un entorno seguro, B tendrá que obtener un $PSO-cert$ de O_1' correspondiente a su entorno de ejecución y así demostrar a A que el objeto resultante O_1' tras la ejecución del servicio también se ha procesado en un entorno seguro. Aquí se pueden encontrar diversos casos donde los PSO-Certs son aplicables. Por ejemplo, si B comparte los actores que han intervenido en la obtención por parte de A del $PSO-cert(O_1)$, entonces el $PSO-cert(O_1)$ en el entorno de B es formalmente idéntico al de A, es decir, $PSO-cert(O_1) = PSO-cert(O_1')$. En el otro extremo, B puede

Cert	Quién lo obtiene	A quién va dirigido	Para qué se usa
$PSO - cert(O_1)$	A (Consumidor)	B(Proveedor)	Ofrece garantías a B de que O_1 ha validado los métodos, el estado y el acceso método-estado en A
$PSO - cert(O'_1)$	B (Proveedor)	A (Consumidor)	Ofrece garantías a A de que el objeto resultante cuyo nuevo estado ha sido obtenido tras la ejecución del servicio, ha sido validado en B
PSC-cert($Servicio^B$)	B (Proveedor)	A (Consumidor)	Ofrece garantías a A de que la implementación del servicio proporcionada por B es segura

Tabla 2.2: Resumen de certificados generados en el escenario de la figura 2.10

tener diferente suministrador, compilador, procesador y verificador en cuyo caso su código compilado, verificado, suministrado y procesado será diferente del que se obtendría en el entorno de A. También hay casos intermedios, tales como por ejemplo que el entorno de B sea compatible con el código compilado por el compilador en A, en cuyo caso no será necesaria transformación alguna por parte de B. En este punto, B, a través de la implementación del servicio puede hacer uso del objeto O_1 y ejecutar sus métodos con el nivel de seguridad que indique $PSO - cert(O_1)$. Como consecuencia, B podrá alterar tanto su propio estado (accedido y/o modificado por la implementación del servicio) como el estado del objeto O_1. En todos los casos, aunque B tuviese que realizar transformaciones de código y/o de estado de O_1, el objeto resultante O'_1 tras el proceso del servicio en B, vendrá acompañado de su correspondiente $PSO - cert$ como garantía de que se ha procesado en un entorno seguro. A modo de resumen, en la tabla 2.2, se muestran los certificados generados en el escenario de la figura 2.10 indicando las garantías obtenidas por su utilización y donde puede observarse que conforme al modelo descrito, A y B comparten estado y proceso cuya seguridad está referida a una política de seguridad basada en certificados de seguridad de código, datos y objetos.

Conformidad al modelo conceptual de seguridad

Los modelos conceptuales de seguridad SOA de código, datos y objetos son modelos conceptuales de referencia puesto que definen un *conjunto mínimo de conceptos y relaciones dentro de un determinado dominio y que es independiente de*

estándares específicos, tecnologías, implementaciones y otros detalles concretos. (MacKenzie et al., 2006). Son modelos conformes al modelo conceptual de seguridad descrito en el apartado 2.3.2 puesto que en efecto aparecen las 5 dimensiones de seguridad indicadas:

1. Los elementos securizables del modelo son, además de los actores intervinientes, el código, los datos y objetos que se intercambian o se usan en una interacción SOA.
2. Las condiciones de seguridad se centran especialmente en los certificados de seguridad (PS-Certs) obtenidos conforme a políticas/contratos de seguridad entre el proveedor y consumidor de un servicio.
3. Las medidas de seguridad son las necesarias para conseguir las condiciones de seguridad establecidas por los actores. El modelo exige al menos las medidas que garanticen la identificación/autenticación y autorización de los actores, el registro de sus actividades en los certificados de seguridad intercambiados y la validación de las operaciones a través de un validador de código, datos y objetos.
4. El modelo propone que todos los actores que afectan o acceden a los elementos securizables deben ser identificados y sus acciones deben ser certificadas. Para ello identifica los actores que son esenciales en el proceso tanto para código como para datos y objetos.
5. Aparecen los tres momentos de aplicación de la seguridad. 1) Prevención: los PS-Certs se obtienen previamente y sirven para determinar la confianza en cada uno de los actores 2) Detección: los PS-Certs pueden ser obtenidos en el mismo momento de la transacción para asegurarse de la fiabilidad del código/datos/objetos que se utilizan y/o transmiten 3) La validación puede ser realizada a posteriori por uno o más validadores para que en caso de que se detecten problemas de seguridad puedan ser corregidos.

2.3.4. Conclusiones

El modelo conceptual de seguridad descrito complementa aquellos modelos conceptuales de seguridad SOA indicados en la sección 2.2.3 con la introducción de la seguridad del código y los certificados de seguridad de código, datos y objetos. Como ya hemos puntualizado, se trata de un modelo conceptual que permite entender mejor la seguridad en un determinado ámbito y cuya principal utilidad

es la de servir de referencia a la hora de abordar un proyecto de seguridad en entornos SOA.

La investigación realizada para la definición de este modelo conceptual nos proporcionó experiencia en la definición de los modelos conceptuales y nos permitió conocer con mayor profundidad los problemas que se producen en la práctica a la hora de aplicar los principales estándares sobre modelos (UML, UML Profiles, MOF), gracias al estudio de la literatura sobre este ámbito. Esto nos motivó para avanzar hacia el siguiente paso: profundizar en los fundamentos de la modelización según la propuesta dominante ofrecida por los estándares OMG. Los resultados de esa investigación serán descritos en los dos capítulos siguientes.

Capítulo 3

Herramienta para el estudio de la modelización. La técnica RCM

En el capítulo anterior se ha mostrado el estado del arte respecto a los modelos de seguridad en general, los modelos de SOA y los modelos de seguridad SOA, tanto desde el punto de vista MDE como no MDE. Igualmente se ha descrito nuestra propuesta de un modelo conceptual que incluye un modelo extendido de SOA para incorporar el aspecto del código y un modelo conceptual de seguridad que amplía los modelos existentes con un enfoque que resalta el papel central que juegan los actores (humanos o no) en la seguridad.

A continuación nuestra intención inicial fue investigar con más profundidad la modelización según los estándares más extendidos propuestos por OMG. Concretamente nos planteamos una investigación en dos pasos:

1. **Comprensión de los conceptos básicos de modelización según OMG**: se trataba de un paso necesario puesto que nuestra especialización procedía del ámbito de la seguridad y nuestro conocimiento de la modelización estaba basado en la experiencia con modelos conceptuales. Respecto a los estándares OMG, únicamente conocíamos los problemas que se describían en la literatura sobre sus dificultades de comprensión y aplicación, tal como hemos indicado en el capítulo anterior.
2. **Estudio de las principales propuestas de metamodelización**: especialmente las relacionadas con MDA (también propuesto por OMG).

En nuestra opinión el primer paso era determinante. No podríamos (como así ocurrió) abordar el segundo paso si no profundizábamos previamente en el conocimiento de los conceptos fundamentales de la modelización con el fin de clarificar

el significado de los conceptos y responder a muchas cuestiones que nos habían surgido en nuestro estudio.

Para la consecución de ese primer objetivo, nuestra investigación se dirigió primero hacia el estudio y comprensión de la bibliografía relacionada con la modelización seleccionando y revisando un extenso conjunto de referencias. Una parte importante de esas referencias tenía que ver con la nomenclatura utilizada. Al comienzo del proceso de revisión nuestra esperanza era que existiera un cierto grado de consenso sobre el verdadero significado de los términos empleados en la modelización puesto que sólo así se podrían evitar ambigüedades en los debates de la literatura y además nos permitiera comprender mejor sus fundamentos. Sin embargo, pudimos comprobar que, por el contrario, los principales debates se centraban precisamente en los aspectos terminológicos. En las referencias bibliográficas sobre este tipo de debates terminológicos nos encontramos con diferentes definiciones de los términos principales de la modelización que en algunos casos eran incluso contradictorias. Algunos de los debates tenían como protagonistas a un grupo de autores que a lo largo de más de una década continúan escribiendo sobre el tema sin que hayan conseguido aún llegar a un consenso sobre el verdadero significado de términos como modelo, metamodelo o lenguaje de modelización (Muller et al., 2009).

El problema de la enorme variedad y cantidad de términos involucrados en estas definiciones nos impedía ver con claridad qué significados eran más relevantes y cuáles eran sus principales relaciones entre ellos. Por ese motivo decidimos buscar una herramienta que nos ayudase en esa labor. Tras analizar las diferentes herramientas existentes (ver sección 3.4) llegamos a la conclusión de que ninguna de ellas era adecuada para nuestros objetivos por lo que decidimos elaborar una herramienta nueva, especializada en el análisis de los conceptos que aparecen en las definiciones de diferentes fuentes bibliográficas que hemos denominado como RCM (References-enriched Concept Map).

Por tanto, antes de continuar con el estudio de los conceptos de modelización, en este capítulo vamos a describir en profundidad los RCM, como construirlos y como utilizarlos para medir la importancia de los conceptos en las definiciones desde el punto de vista de las diferentes referencias bibliográficas. Como veremos en el capítulo 4, los RCM aplicados a la modelización nos permitirán comprender mejor las definiciones debatidas en la bibliografía.

Una primera versión de los RCM fue presentada en el Congreso MODELS 2010

en un trabajo sobre problemas de la modelización (Rodriguez-Priego et al., 2010). Posteriormente, con un ligero cambio de nombre, su descripción completa fue publicada en la revista Journal of Information Science (Rodriguez-Priego et al., 2013).

3.1. Definiciones

3.1.1. Definición (de *definición*)

Antes de explicar qué es un RCM es necesario entender un poco mejor qué es una *definición*.

Definición **3.1.** Definición
Según la Real Academia de la Lengua, *definición* se define como:

> Proposición que expone con claridad y exactitud los caracteres genéricos y diferenciales de algo material o inmaterial.

Desde un punto de vista clásico (Granger, 1984) podemos decir que las definiciones, sean léxicas, por convenio o aclaratorias (Gupta, 2012) se componen típicamente de dos elementos:

- **Definiendum**: (plural *definienda*) es el término que va a ser definido
- **Definiens**: (plural *definientia*) es la expresión que explica el significado del *definiendum*. A su vez *definiens* está compuesto habitualmente por dos elementos:
  1. **Genus**: es la *esencia* del concepto, es decir, la clase o familia a la que pertenece el Definiendum
  2. **Differentia**: es el conjunto de características que lo diferencian de otros conceptos de la misma familia

3.1.2. Propiedades de las *buenas* definiciones

Diversos autores (Gupta, 2012; Hurley, 2011; Joyce, 1916) han llevado a cabo diferentes clasificaciones de definiciones y han analizado las características de una *buena definición*, especialmente para el caso de las definiciones por genus y differentia y las definiciones léxicas. En opinión de estos autores, las características que debe cumplir una buena definición son principalmente las siguientes:

- **Debe mostrar los atributos esenciales de *definiendum***: tanto en el *genus* como en el *differentia* deben resaltarse los conceptos principales.
- **No debe ser circular**: el *definiendum* no debe aparecer en el *definiens*.
- **No debe ser demasiado genérica**: la *differentia* debe ser suficiente para que el *definiendum* no pueda ser confundido por otros *definienda*. Por ejemplo, decir que una ballena es solo un mamífero acuático es demasiado genérico puesto que los delfines también son mamíferos acuáticos.
- **No debe ser demasiado específica**: la *differentia* no debe ser tan restrictiva que excluya a algunas entidades a las que se refiere. Por ejemplo, decir que un cisne es un ave de color blanco es demasiado específico puesto que hay cisnes de color negro.
- **No debe ser confusa**: es decir, no debe ser expresada con ambigüedad o con un lenguaje figurativo o difícil de entender.
- **Debería evitar una terminología *afectiva***: es decir, no deben incluirse frases o términos que jueguen con los sentimientos o creencias del lector.
- **Debería indicar el contexto en el que se realiza la definición**: puesto que de no hacerlo podría no entenderse a qué se está refiriendo.
- **Debe expresarse en términos afirmativos**: es preferible definir, siempre que sea posible, lo que es el *definiendum* en lugar de lo que no es.

3.2. *Concept Maps*

Puesto que RCM se basa en *mapas conceptuales* o *concept maps*[1], a continuación describimos brevemente como referencia qué son y cómo se utilizan según los trabajos de Cañas et al. (2005); Novak and Gowin (1984); Novak and Cañas (2008, 2006).

3.2.1. Definición de *Concept Maps*

Definición **3.2.** Concept map
Es una herramienta gráfica para organizar y representar el conocimiento. Los conceptos son representados mediante círculos o cajas que contienen el nombre

[1]En adelante utilizaremos predominantemente los términos originales en inglés, especialmente en aquellos casos en los que el término en español es menos utilizado o pueda resultar ambiguo.

del concepto (que puede ser una o más palabras). Las relaciones entre conceptos se representan mediante líneas de conexión entre ellos y pueden ser nominadas. Las relaciones con nombre, denominadas *linking phrases*, se indican mediante frases sobre las líneas de conexión.(Novak and Cañas, 2008).

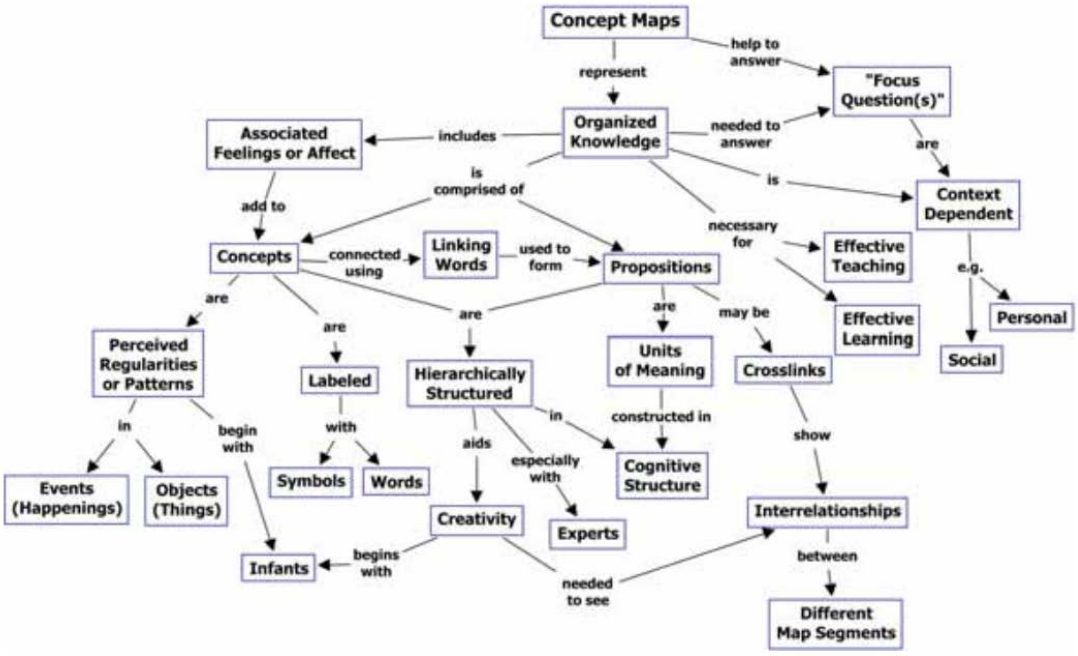


Figura 3.1: *Concept map* representando las principales características de los *concept maps* (tomado de (Novak and Cañas, 2008))

3.2.2. Características de los *Concept Maps*

Veamos algunas de las características más relevantes de los *concept maps* según (Novak and Cañas, 2008):

- **Jerarquía top-down**: los conceptos se representan jerárquicamente desde los más generales (generalmente en la parte superior del gráfico) hasta los más específicos de manera descendente. En el ejemplo de la figura 3.1, tomada de (Novak and Cañas, 2008), puede observarse cómo efectivamente los conceptos se representan de manera jerárquica.
- **Contexto**: la situación o evento que se pretende comprender a través de la organización del conocimiento se denomina *contexto* del *concept map*. En el ejemplo de la figura 3.1, el contexto es la comprensión de los propios *concept maps*.

- **Focus question**: los *concept maps* se construyen generalmente tomando como referencia alguna pregunta, relativa al contexto, de la que se pretende encontrar respuesta y que se denomina *focus question*. Las *focus questions* ayudan a delimitar el contexto en el que se construye un *concept map*. En el ejemplo la *focus question* es ¿qué son los *concept maps*?
- **Propositions**: las expresiones que contienen dos o más conceptos conectados mediante *linking phrases* para formar una sentencia con significado se denominan *propositions*. Ejemplos de proposiciones encontradas en la figura 3.1 son: *Concept Maps represent Organized Knowledge*, *Propositions may be Crosslinks*, *Concepts are Labeled with Symbols*, etc.
- **Cross-links**: las relaciones o enlaces entre conceptos de diferentes segmentos o dominios de un *concept map* se denominan *cross-links* y ayudan a ver cómo un determinado concepto de un determinado dominio del conocimiento que esté representado en el *concept map* se relaciona con otros dominios de conocimiento que también son representados. El pensamiento creativo (*creative thinking*) viene facilitado por dos características de los *concept maps*: la estructura jerárquica y la habilidad para encontrar y caracterizar nuevos *cross-links*. En el ejemplo de la figura 3.1, podemos ver que *needed to see* es un *cross-link* entre los conceptos *Creativity* y *Interrelationships*.
- **Ejemplos**: los *concept maps* pueden incluir ejemplos específicos que ayuden a aclarar el significado de un determinado concepto (no deben representarse dentro de cajas o óvalos para no confundirlos con los conceptos)
- **Enlaces**: mediante la utilización de software especializado[2] para dibujar *concept maps* es posible incluir enlaces a fotos, gráficos, videos, páginas web, etc., asociándolos a un concepto o a una *linking phrase*.
- **Compartición**: algunas herramientas permiten la compartición de *concept maps* a través de servidores públicos e incluso la creación de *concept maps* de manera colaborativa.

3.2.3. Método de elaboración de los *Concept Maps*

Según Novak and Cañas (2008) los principales pasos para construir *buenos concept maps* son los siguientes:

[2]En la figura 3.4.4 se indica una lista de software especializado para *concept maps* y otras técnicas de organización y representación del conocimiento.

1. **Determinar el contexto** y la *focus question* del *concept map*: de tal manera que el *concept map* ayude a responder a la pregunta de referencia (*focus question*).
2. **Identificar los conceptos** clave relacionados con ese dominio (entre 15 y 25 conceptos deberían ser suficientes).
3. **Ordenar los conceptos** desde el más general al más específico. Esta lista, conocida como *parking lot*, será sólo una primera aproximación al problema y servirá de base para construir el *concept map* (algunos conceptos de esa lista podrían no utilizarse si finalmente no pueden ser ubicados en el diagrama final).
4. **Construir una versión preliminar** del *concept map* a partir del *parking lot* colocando los conceptos más generales en la parte superior y uniendo ordenadamente los conceptos relacionados mediante *linking phrases*, usando preferiblemente software especializado. Debe evitarse incluir frases completas como conceptos en la medida de lo posible. También debe evitarse la construcción del *concept map* mediante el encadenamiento de conceptos secuencial, denominado *string map*.
5. **Identificar *cross-links*** entre conceptos de diferentes segmentos o subdominios dentro del *concept map*.
6. **Revisar la versión preliminar** del *concept map* para añadir nuevos conceptos, *linking phrases* o *cross-links*. Normalmente se necesitan tres o más revisiones para obtener una versión *definitiva* (aunque Novak and Cañas indican que la elaboración de un *concept map* es un proceso que en realidad nunca finaliza).

Por otra parte, (Cañas et al., 2005) también indican algunas características adicionales de los *buenos concept maps*:

1. Cada *par* de conceptos, junto con su *linking phrase*, debe poder ser leída como una sentencia individual o proposición con sentido.
2. Los conceptos y las *linking phrases* deben ser tan cortas como sea posible, generalmente palabras individuales. Generalmente los conceptos son sustantivos o sintagmas nominales, mientras que las *linking phrases* son verbos o sintagmas verbales.
3. La estructura debe ser jerárquica y el nodo raíz del mapa debe ser representativo de la *focus question* del *concept map*.

Obsérvese que la restricción de que cada par de conceptos debe formar una sentencia con significado del punto 1, entra en contradicción con la definición de *Propositions* indicada por (Novak and Cañas, 2008) y a la que hicimos referencia en la sección 3.2.2 donde se afirma que las sentencias con significado pueden estar formadas por dos o más conceptos. En nuestra opinión, la definición correcta es la de la sección 3.2.2 puesto que una proposición puede estar formada por *más de dos conceptos*[3]. Así por ejemplo, en la figura 3.1, también incluida en (Cañas et al., 2005), podemos observar que la relación *Interrelationship between Different Map Segments* no tiene sentido por sí misma, mientras que sí lo tiene con un concepto y enlace más: *Crosslinks show Interrelationship between Different Map Segments*.

3.3. References-enriched Concept Maps

Tomando como referencia los apartados anteriores sobre definiciones y *concept maps*, en esta sección vamos a definir qué son los RCM, sus características, ejemplos, usos, el método de elaboración así como un conjunto de métricas que nos permitirán realizar análisis sobre ellos.

3.3.1. Definición de RCM

Definición **3.3.** References-enriched Concept Map (RCM)
RCM es una técnica basada en *concept map* que permite la visualización simultánea de un conjunto de definiciones sobre un término que han sido discutidas en la literatura, facilitando el análisis de tales definiciones en relación con los autores que las propusieron. El *definiendum* es el *concepto principal* del RCM y los *definientia* son las diferentes definiciones cuyos conceptos son representados como en un *concept map* pero añadiendo *labeled paths* que permiten la reconstrucción de las definiciones de cada referencia.

3.3.2. Características de los RCM

En este apartado describiremos las características de los RCM.

[3]Curiosamente uno de los autores del punto 1 también es autor de las propiedades indicadas en la sección 3.2.2

- **Focus question**: el *contexto* de un RCM habitualmente es el correspondiente al significado del *definiendum*, siendo la *focus question* del tipo *¿cuál es el significado de* definiendum *en la literatura?*.
- **Path labeling** con referencias: una de las principales características de un RCM es el etiquetado, no ya de conceptos o *linking phrases* individuales, sino de rutas completas (paths) que abarcan varios conceptos y los enlaces entre ellos (proposiciones) con el fin de hacer explícita la autoría de las proposiciones. Es lo que denominamos *path labeling*. De este modo, las proposiciones atravesadas por una determinada ruta (path) son etiquetadas mediante una referencia que informa sobre su responsable (por ejemplo, quién es el autor de tal proposición). Para facilitar la legibilidad del diagrama cada referencia se representa por una etiqueta utilizando preferentemente algún estilo bibliográfico estándar (ACM, APA, etc.). Un lector del RCM podrá reconstruir cada contribución contenida en él, sin más que seguir la ruta marcada por cada referencia, incluso cuando una ruta está incluida en otra (veremos un ejemplo en el apartado 3.3.3). Nótese que esta técnica es propia de los RCM y permite que éstos puedan ser construidos por un tercero diferente a los autores originales de las proposiciones recogidas. Así ocurre cuando un investigador confecciona un RCM recogiendo las propuestas que varios autores realizan sobre un tema en diferentes trabajos. Mediante el *path labeling*, el creador del RCM puede atribuir el autor a cada proposición por él reflejada en el mapa, usando para ello referencias a cada una de las publicaciones consultadas. Es precisamente este uso del *path labeling* el que originó los RCM. Por este motivo, *path labeling* es una importante contribución de los RCM respecto a los *concept maps*.
- **Colaboración**: como hemos indicado en el apartado 3.2.2, en ocasiones los *concept maps* pueden ser compartidos e incluso creados de manera colaborativa. En ese caso cada colaborador del *concept map* puede proporcionar parte de su conocimiento. Sin embargo, el *concept map* resultante no refleja gráficamente quién es el responsable de cada contribución individual. Por el contrario, en un RCM la contribución de cada colaborador sí queda reflejada de manera explícita gracias al *path labeling*, puesto que cualquier usuario del RCM puede identificar qué parte del mismo corresponde a cada uno de los autores que han colaborado en su elaboración.
- **Cross-links**: el orden indicado por las referencias no impide el estable-

cimiento de *cross-links* entre los conceptos del RCM dado que éste sigue siendo un *concept map* pero enriquecido con conocimientos sobre el *concepto principal*. Los *cross-links*, como en *concept maps* pueden establecerse entre segmentos o subdominios del RCM pero también entre RCM, lo que permite establecer relaciones entre conceptos de diferentes definiciones (esta característica también es tenida en cuenta en el análisis de las métricas que se describe en el apartado 3.3.8).

- **Concept layering**: RCM incluye una característica que ayuda a cada creador del RCM a resaltar la importancia de cada concepto en una definición. En el apartado 3.1.1 hemos visto que tras el *definiendum* suele seguir el *genus*. Generalmente el *genus*, o en general el concepto que aparece tras alguna forma del verbo *ser*, definen la esencia de lo que hemos denominado *concepto principal*. RCM permite resaltar aquellos conceptos que en opinión del creador del RCM (que puede ser el autor directamente o un tercero que interpreta a un autor) son situados en un primer nivel más cercano a la esencia del *concepto principal*. Esto permite distinguir a esos conceptos, que denominaremos *conceptos primarios*, de los conceptos que proporcionan características descriptivas o aclaratorias. Denominaremos *niveles* a los niveles resultado de esta clasificación de conceptos y los nombraremos consecutivamente *primario, secundario, terciario, etc.* ordenados de mayor a menor importancia. El creador del RCM puede también clasificar este resto de conceptos en otros niveles, aunque lógicamente por razones de legibilidad es conveniente no utilizar más de tres o cuatro niveles. Debe tenerse en cuenta que la clasificación en niveles es una decisión del creador del RCM y no tiene por qué coincidir con los niveles jerárquicos de un *concept map*. De hecho esa jerarquía es uno más entre los posibles criterios que puede adoptar el creador del RCM. Sin embargo, lo habitual, en base a lo que se ha descrito, es que los conceptos del *genus* se sitúen en el *nivel primario*, los de la *differentia* en el *nivel secundario* y el resto de conceptos en los siguientes.

  Veamos un ejemplo. En la definición “una mesa es una pieza de mobiliario con una superficie plana soportada por una o más patas”, “mesa” es el *definiendum*, “pieza de mobiliario con una superficie plana soportada por una o más patas” es el *definiens*, “pieza de mobiliario” es el *genus* y “con una superficie plana soportada por una o más patas” es la *differentia*.

Desde el punto de vista de RCM, "mesa" es el *concepto principal* y probablemente el creador del RCM elegirá "pieza de mobiliario" como un *concepto primario* o perteneciente al *nivel primario* y "superficie plana" y "patas" como *conceptos secundarios* o pertenecientes al *nivel secundario*.

Para representar los niveles, RCM usa un determinado color con densidad variable, asignándose un color más denso al *nivel primario* y disminuyendo progresivamente con el nivel. El *concepto principal* se representa con el color sólido. De este modo los *usuarios* de un RCM pueden, mediante una visión rápida, diferenciar cuáles son los conceptos más importantes en una definición.

Es posible que, en diferentes definiciones, un mismo concepto aparezca de forma natural en diferentes *niveles*. En estos casos, el creador del RCM debe elegir el nivel definitivo que asignará al concepto en el RCM. Para ello podría basarse en el número de referencias que usan el concepto en uno u otro *nivel*.

En la tabla 3.1 se muestra una comparativa resumen entre las propiedades de los *concept maps* y los RCM.

Concept Maps	**References-enriched Concept Maps**
Conceptos	El *definiendum* y los conceptos incluidos en los *definientia*
Linking phrases	*Linking phrases*, la primera de las cuales debería ser alguna forma del verbo "ser"
Proposition	Definición o parte de una definición
Focus question	¿Cuál es el significado de *concepto principal*?
Contexto	Discusión sobre el significado de un término
Cross-links	Permiten la comparación entre diferentes definiciones (*definientia*) del mismo concepto (*definiendum*) y la comparación entre definiciones de diferentes conceptos (*definienda*)

Tabla 3.1: Correspondencia entre concept maps y RCM

3.3.3. Ejemplo básico de RCM

Veamos un sencillo ejemplo que nos permita conocer el aspecto de un RCM típico y los elementos que lo componen según se ha descrito anteriormente. La figura 3.2 muestra una versión reducida del RCM de *Language* (la versión completa será mostrada en el apartado 4.1.3). El RCM se ha construido a partir de cuatro

definiciones obtenidas de cuatro referencias que también se muestran en la figura 3.2:

- *Language is a concept*
- *Language is a set of expressions used to communicate*
- *Language is a set of sentences and has specific rules*
- *Language is a set of sentences constructed from a finite set of elements*

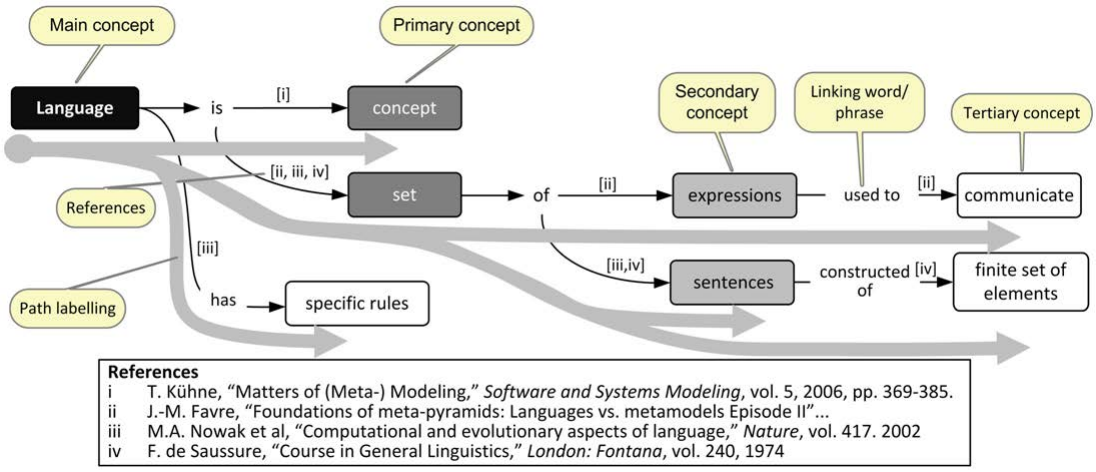


Figura 3.2: Ejemplo básico de RCM

Para el *concept layering* hemos elegido el color negro, representando al *concepto principal* con color sólido. Como puede observarse hemos organizado los conceptos de los *definientia* en tres *niveles* fácilmente identificables por los colores gris oscuro (*primario*), gris (*secundario*) y blanco (*terciario*). Obsérvese que los niveles no se corresponden directamente con los de una jerarquía de *concept maps*. Efectivamente, aunque el concepto *specific rules* aparece en la definición tras el concepto *sentences*, hemos interpretado como creadores del RCM, que se trata de un concepto descriptivo que no forma parte del *genus* de la definición. En la figura se han añadido flechas de color gris que no forman parte de la sintaxis de RCM pero que permiten mostrar gráficamente el *path labeling* de este RCM.

Por otra parte, obsérvese que el *path labeling* permite que una definición esté contenida en otra (referencias iii y iv) y aún así poder seguir identificando las referencias a las que pertenecen. Sin el *path labeling* no sería posible distinguir la autoría de cada una de ellas.

3.3.4. Autoría y roles en un RCM

Como se ha visto, los RCM ofrecen mediante el *path labeling* la característica necesaria para realizar mapas conceptuales que sean efectivamente colaborativos.

Para entender mejor el uso de esta característica, podemos distinguir los siguientes roles:

1. **Creador**: elabora un RCM creando nuevas proposiciones por sí mismo o recolectando e interpretando proposiciones atribuibles a los autores de un conjunto de referencias.

2. **Usuario**: usa los RCM para algún fin.

Una misma persona puede ejercer cualquiera de los dos roles simultáneamente y un rol puede ser ejercido por varias personas. Por tanto, cuando utilizamos el término genérico *creador del RCM* en realidad nos estamos refiriendo a un rol que puede ser ejercido simultáneamente o no, por una o varias personas.

Teniendo en cuenta los roles descritos podemos distinguir tres escenarios típicos de uso de los RCM:

1. **Investigador autónomo**: un *creador* analiza la literatura sobre un determinado tema y recopila definiciones realizadas por los autores de unas determinadas referencias para la elaboración del RCM.

2. **Elaboración colaborativa**: varios *creadores* realizan un RCM de manera colaborativa elaborándolo por sí mismos sin la utilización de referencias de la literatura. Para ellos podrán utilizar herramientas basadas en web que permitan la edición simultánea de RCM. El origen de cada definición queda reflejada automáticamente a través de *path labeling*.

3. **Creación colaborativa mixta**: varios *creadores* realizan un RCM de forma colaborativa utilizando tanto definiciones propias como interpretando las definiciones de otros autores encontradas en un conjunto de referencias bibliográficas.

En cualquiera de los casos, podemos ver un RCM como la *fusión* de varios *concept maps* (uno por cada *creador*) (puede imaginarse una superposición) haciendo coincidir conceptos y *linking phrases* y en la que la autoría siempre puede ser identificada.

3.3.5. Método de elaboración de RCM

El procedimiento para la construcción de un RCM es diferente del descrito por Novak and Cañas y que resumimos en el apartado 3.2.3. Efectivamente Novak and Cañas sugieren recopilar los conceptos en una lista y clasificarlos de mayor a menor generalidad para luego situarlos donde el creador del *concept map* estime más conveniente. Sin embargo, en un RCM no tenemos siempre la misma libertad, sobre todo en el caso de un escenario de *investigador autónomo*, donde el orden de los conceptos en cada una de las definiciones viene determinado por el que han decidido los autores de las mismas. Aun así, como veremos a continuación, el *creador* del RCM tiene algo de libertad para reformular y consolidar conceptos y *linking phrases*.

En general, los pasos que han de seguirse para la construcción de un RCM en ese caso son los siguientes:

1. **Recopilación de las referencias**: realizar una lista de las referencias que contienen definiciones del *concepto principal*.
2. **Identificación y reformulación de definiciones**: identificar cada una de las definiciones contenida en cada referencia resaltando los conceptos y las *linking phrases*. En algunas ocasiones será necesario reformular las definiciones para obtener una sentencia equivalente con el fin de facilitar la identificación y consolidación de los conceptos comunes entre ellas. Así por ejemplo, si una referencia indica que *languages are symbol systems* y otra referencia que *language is a system of symbols* debería entenderse que ambas son equivalentes, o lo que es lo mismo, se corresponden con los mismos conceptos y *linking phrases*. En este caso, sería preferible usar el orden y conectivas obtenidos directamente de la última frase.
3. **Disposición de conceptos y *linking phrases***: enlazar los conceptos mediante líneas dirigidas mediante flechas y etiquetadas por *linking phrases*, de manera similar a como se construiría un *concept map*, aunque con las limitaciones que establecen las propias definiciones y disponiéndolos habitualmente de izquierda a derecha en lugar de arriba a abajo.
4. **Path labeling**: etiquetar las líneas que suceden a las *linking phrases* con la referencia donde aparezca cada definición. La etiqueta debe ser única por definición y en el caso de que una misma referencia contuviese dos o más definiciones donde pudiera producirse ambigüedad respecto a su *path*,

deberá usarse una etiqueta diferente como por ejemplo, 14a y 14b (para un ejemplo de este caso, véase más adelante en la figura 4.4 el RCM de modelo). El resultado final es que siguiendo el *path* de cada referencia deben poder obtenerse de una forma unívoca cada una de las definiciones sin posibilidad de confusión entre ellas.

5. **Concept layering**: elegir un criterio para la definición de niveles y clasificar los conceptos en cada nivel tal como se ha explicado en el apartado 3.3.2.

Siguiendo estos pasos pueden construirse RCM de complejidad elevada pero formando un diagrama compacto, como por ejemplo el RCM de *model* que está compuesto por 26 definiciones, cuya primera versión fue presentada en el Congreso MODELS 2010 (Rodriguez-Priego et al., 2010). Éste y otros RCM de cierta complejidad serán descritos en el capítulo 4.

A pesar de que un RCM puede ser realizado con lápiz y papel, existen diversas aplicaciones informáticas para la construcción de *concept maps* que, con algunas limitaciones, pueden usarse para la creación de RCM. En el apartado 3.4 mencionaremos algunas de esas herramientas y su idoneidad para la creación de RCM.

Por otra parte, debe tenerse en cuenta que un *concept map*, y por lo tanto un RCM, es generalmente una aproximación de trabajo a la representación de los conceptos y relaciones en un determinado dominio de conocimiento (Novak and Gowin, 1984). En el caso particular de los RCM esto implica que por un lado el creador del RCM debe seleccionar las definiciones que considera relevantes entre las que encuentra disponibles en el contexto de su estudio y, por otro lado, que los RCM siempre están vinculados a la fecha de la última actualización, la cual viene determinada por la fecha de la referencia más reciente. Por tanto, un RCM está sujeto a nuevas versiones derivadas de las actualizaciones que realice el creador del RCM a medida que vayan apareciendo nuevas definiciones de nuevas referencias.

3.3.6. Usos de los RCM

Como hemos indicado, el principal objetivo de los RCM es proporcionar un apoyo en la investigación y clarificación del significado de los conceptos más relevantes en un determinado contexto. Más concretamente, a continuación se relacionan las situaciones en las que su utilización resulta más útil:

- **Comparación de definiciones**: los RCM son particularmente útiles como

herramienta para comparar definiciones, sobre todo en aquellos casos donde no existe consenso en la literatura sobre la terminología, como es el caso de la modelización, que fue la motivación para su desarrollo.

- **Definiciones complejas**: ayudan a visualizar de una manera más clara los conceptos más importantes. Esto es especialmente útil en las definiciones complejas en las que intervienen muchos conceptos y relaciones.
- **Visualización compacta**: es habitual encontrar listados de las principales definiciones en aquellos artículos que estudian una determinada materia. Un RCM es una alternativa visual mucho más compacta sobre todo en definiciones complejas, puesto que permite la representación de muchas definiciones usando menos espacio.
- **Análisis de referencias bibliográficas**: mediante métricas (que serán descritas en el apartado 3.3.8), los RCM permite evaluar entre otros aspectos, la calidad o la similitud de las definiciones e indirectamente de las referencias que las contienen.
- **Definiciones resumen**: una aplicación muy interesante de los RCM es la posibilidad de construir *definiciones resumen*, es decir definiciones construidas a partir de *todas* las definiciones no contradictorias contenidas en el RCM. Este tipo de definiciones ofrecen una foto global sobre el *estado del arte* en lo que respecta a una determinada terminología. En el apartado 6.8 presentaremos una definición de este tipo de definiciones extraida del RCM de modelo.

3.3.7. Generalización de los RCM a otros contextos

Aunque la aplicación original de los RCM es el de las definiciones en un escenario de investigador autónomo, sus dos principales características diferenciadoras, *path labeling* y *concept layering*, son lo suficientemente genéricas para permitir que los RCM puedan ser aplicados en otros contextos y escenarios. Así por ejemplo, Garcia (2011) de Technische Universiteit Eindhoven aplicó la técnica RCM para la representación de los elementos, objetivos y soluciones de la modelización de procesos.

En un contexto general y en un escenario colaborativo, un grupo de creadores podrá aplicar el método descrito en el apartado 3.3.5 pero considerando que una definición es un tipo concreto de proposición y todo lo dicho acerca de las

definiciones, salvo sus particularidades como su estructura de genus y differentia, puede aplicarse a cualquier tipo de proposición.

Por tanto, los RCM pueden ser considerados como una técnica que mejora la de los *concept maps* facilitando la colaboración entre creadores de RCM (mediante *path labeling*) y la visualización de la relevancia de los conceptos (mediante *concept layering*) en base a diferentes criterios.

3.3.8. Métricas para el análisis basado en RCM

Los *concept maps* no sólo sirven como herramienta para la representación del conocimiento sino que de su análisis es también posible obtener conclusiones sobre diferentes aspectos del aprendizaje (Marchand et al., 2002; Novak and Gowin, 1984). Del mismo modo, tal como hemos indicado en el apartado 3.3.6 uno de los usos de los RCM es el del análisis del conocimiento, especialmente en lo relativo a la calidad, complejidad, etc. del RCM. Así por ejemplo, el grado de complejidad de un RCM es proporcional a varias de las métricas que se describen a continuación como la dispersión conceptual y semántica.

En este apartado describiremos métricas que permiten al *usuario* del RCM realizar el análisis de los conceptos, sus relaciones, dependencias, etc. en los contextos asociados a uno o varios RCM. La tabla 3.2 muestra un resumen de las métricas que se describen a continuación, indicándose para cada métrica el ámbito sobre el que se aplica (concepto, campo semántico, nivel, definición, RCM), el criterio aplicado (Marchand et al., 2002; Talbi et al., 2001), su tipo (ratio o porcentaje) y finalmente se indica si la métrica proviene de propiedades del RCM que se evalúan objetiva o subjetivamente.

Debe tenerse en cuenta que la aplicación de las métricas que van a describirse a continuación tiene sentido únicamente cuando están referidas a RCM de cierta complejidad, puesto que aplicadas a RCM simples (como el del apartado 3.3.3) se obtendrían resultados triviales. Puesto que en el capítulo 4 se aplica la técnica RCM para el análisis de la terminología de la modelización, obteniéndose varios RCM de mayor complejidad, encontraremos en ese capítulo ejemplos de aplicación de estas métricas que nos permitirán obtener conclusiones interesantes sobre los conceptos de modelización usados en la literatura.

Métrica	Ámbito	Criterio	Tipo	Subjetivo
Dispersión semántica y conceptual	Nivel	Mismo *campo semántico*	Ratio	No
Densidad de referencias	Concepto, Campo semántico	Porcentaje de referencias	Porcentaje	No
Calidad	Definición	Relevancia, Circularidad, Generalidad y Especifidad, Claridad, Asertividad	Porcentaje	Sí
Similitud	Definición	Media del porcentaje de conceptos comunes en definiciones	Porcentaje	Sí
Dependencias	Definición	Inclusión de conceptos en referencias; Relevancia de conceptos	Ratio	No
Contradicciones e inconsistencias	Definición	Detección de conflictos entre definiciones; Contradicciones lógicas	Ratio	Sí
Similitudes	RCM	Sinonimia, Antonimia, Inconsistencias	Ratio	Sí

Tabla 3.2: Resumen de métricas sobre RCM

Dispersión conceptual y semántica

Las siguientes métricas permiten evaluar el grado de dispersión que existen entre conceptos o grupos de conceptos en un RCM.

Definición **3.4.** Dispersión semántica
Si denominamos *campos semánticos* a una agrupación de los conceptos sinónimos dentro de un mismo *nivel* en un RCM, se define Dispersión semántica (*semantic scattering*) SS_n del *nivel* n como el ratio:

$$SS_n = 100\frac{SF_n - 1}{D - 1}$$

siendo SF_n el número de *campos semánticos* en el *nivel* n y D el número de definiciones en el RCM.

Esta métrica es inversamente proporcional al grado de consenso sobre el significado del *concepto principal* en la literatura. Así por ejemplo cuanto mayor es el valor de SS_1 menos acuerdo ha habido en las definiciones contenidas en el RCM.

La *Dispersión semántica Global* (*Global semantic scattering*) GSS se calcula como el valor medio de la *dispersión semántica* para todos los *niveles*.

Definición **3.5.** Dispersión conceptual
Se define *Dispersión conceptual* (*Concept scattering*) CS_n del *nivel* n como el ratio:

$$CS_n = 100\frac{NC_n - 1}{D - 1}$$

siendo NC_n el número de conceptos en el *nivel n*.

Como mostraremos más adelante, esta métrica es un indicativo de la complejidad de un RCM y es complementaria a SF_n. Así por ejemplo no sería lo mismo SS=10 % con un NC del 10 o con el 90 %. En el segundo caso indicaría que los *campos semánticos* contienen muchos conceptos o lo que es lo mismo, que se utilizan muchos sinónimos en ese *nivel*.

Del mismo modo, la *Dispersión conceptual global* (*Global concept scattering*) GCS se calcula como el valor medio del *dispersión conceptual* para todos los *niveles*.

Tanto SS_n como CS_n nos ofrecen una medida de la dispersión que existe entre los conceptos manejados en las diferentes definiciones sobre un mismo *concepto principal*. Aunque ambos son interesantes, SS_n nos proporciona una medida más aproximada de la dispersión desde el punto de vista semántico. Así por ejemplo si sólo se considerase la dispersión conceptual, un conjunto de definiciones donde se empleasen sinónimos daría la falsa impresión de que existe mucha discrepancia entre los autores acerca del significado del *concepto principal*.

Densidad de referencias

Definición **3.6.** Densidad de referencias por concepto
Se define *Densidad de referencias* (*Density of references*) del concepto c en su *nivel* n, $DRC_n(c)$ como el siguiente porcentaje:

$$DRC_n(c) = 100\frac{r_n(c)}{R}$$

siendo $r_n(c)$ el número de referencias que llegan al concepto c en el nivel n y R el total de referencias del RCM

Definición **3.7.** Densidad de referencias por campo semántico
Se define *Densidad de referencias* (*density of references*) del *campo semántico* sf en el *nivel* n, $DRSF_n(sf)$ como el siguiente porcentaje:

$$DRSF_n(sf) = 100\frac{r_n(sf)}{R}$$

siendo $r_n(sf)$ el número de referencias que llegan al *campo semántico* sf en el nivel n.

Calidad de definiciones

Tomando como referencia las principales propiedades de las definiciones, descritas en el apartado 3.1.2 se definen las siguientes métricas relacionadas con su calidad:

- De la característica *Debe mostrar los atributos esenciales del definiendum* podemos deducir una métrica de esencialidad. El problema es determinar de una manera cuantitativa cuáles son los atributos esenciales de una determinada definición. Nuestra propuesta es asociar un factor de relevancia a cada uno de los conceptos de una definición y determinar la métrica *Definition essentiality* DE como la media ponderada de la esencialidad de sus conceptos ponderados por *nivel*. El criterio elegido para la determinación de la esencialidad de cada concepto es su *Density of references* descrita en la Definición 3.6, de tal manera que:

  DEFINICIÓN **3.8.** Relevancia de un concepto
  Se define *Relevancia de un concepto* c (*Concept relevance*), como:

  $$CR(c) = \frac{DRC_n(c)*(N-n+1)}{\sum_{i=1}^{N} i}$$

  donde N es el número de *niveles* del RCM y n el *nivel* al que pertenece el concepto c.

  DEFINICIÓN **3.9.** Esencialidad de una definición
  Se define *Esencialidad* de una definición d (*Definition essentiality*), como:

  $$DE(d) = \sum_{c\ in\ d} CR(c)$$

  donde c es cada uno de los conceptos que aparecen en la definición d.

- Respecto a la característica *no debe ser circular*, RCM puede ayudarnos a detectar circularidades en una definición. Para conseguir ese objetivo, deberíamos obtener los RCM de cada uno de los *conceptos primarios* de la definición y analizarlos recursivamente para detectar si vuelven a estar relacionados con el *concepto principal*.

- Las características *No debe ser demasiado genérica* y *No debe ser demasiado específica* implican la coexistencia en un RCM de conceptos generales y específicos en el mismo *nivel*, lo cual es probablemente un síntoma de que algunas de las definiciones están excluyendo a casos en los que la definición sería aplicable o incluyendo a otros que realmente no son aplicables. En general se trata de propiedades difíciles de evaluar. A pesar de ello podemos obtener una medida de la generalidad si se analizan conjuntamente un número suficiente de RCM de un determinado dominio de conocimiento. Por ejemplo, dada la definición *el hombre es un ser vivo*, podemos deducir que es general si la comparásemos con otros RCM de otros seres vivos como

gato, perro o árbol. Estos RCM incluirán probablemente esta proposición entre sus definiciones en cuyo caso queda claro que puesto que se trata de seres diferentes, esa generalidad no permite distinguirlos. Por el contrario la definición alternativa *el hombre es un ser vivo con inteligencia* no estaría incluida en las definiciones de los otros seres vivos.

- La característica *No debe ser confusa* puede ser analizada en un RCM identificando la *claridad* de los conceptos y las *linking phrases* empleados en la definición. Se trata de una métrica subjetiva que puede evaluarse como el porcentaje de los conceptos considerados confusos en una definición, ponderados por *nivel*.
- Finalmente podemos evaluar la característica *debe expresarse en términos afirmativos* estudiando las *linking phrases* conectadas directamente al *concepto principal* identificando formas negativas de los verbos.

Similitudes en definiciones

Un RCM permite la comparación de las definiciones *nivel* a *nivel*, resaltando aquellas que son similares mediante el criterio del número de conceptos que comparten. Este análisis nos permite obtener una medida de la similitud entre ellas.

Definición **3.10.** Similitud de definiciones
Se define *Similitud de una definición* d (*Similarity of definition*) como

$$SD_d = \frac{\sum_{\forall d' \neq d} SD(d,d')}{D-1}$$

siendo $SD(d, d')$ la similitud entre las definiciones d y d' definida como

$$SD(d, d') = 100 * \frac{CC(d,d')}{max(d,d')}$$

donde CC(d,d') es el número de conceptos que comparten d y d' y max(d,d') el valor máximo del número de conceptos de d y el número de conceptos de d'

Dependencias entre definiciones

Finalmente los RCM nos permiten encontrar conexiones entre sus definiciones, como por ejemplo que las referencias en un cierto *nivel* para un determinado concepto sean un subconjunto de las referencias de otro concepto en el mismo *nivel* (lo cual también permitiría determinar la relevancia de un concepto respecto a sus referencias) . Se trata de una métrica que mide el número de esas relaciones respecto al número de definiciones D.

Contradicciones e inconsistencias

Un RCM permite detectar y medir el número de contradicciones entre las definiciones analizando y comparando los conceptos de un determinado *nivel* para comprobar si existen conceptos opuestos (teniendo en cuenta lógicamente la posible existencia de *linking phrases* negativas).

Por otra parte también pueden detectarse y medirse la existencia de inconsistencias cuando las definiciones nos llevan a contradicciones lógicas, considerando la métrica *inconsistency ratio* CI de un RCM como el número de definiciones con contradicciones o inconsistencias respecto a D.

Similitud entre RCM

Además de las similitudes entre definiciones en un mismo RCM, es interesante encontrar similitudes entre las definiciones (y por tanto entre *conceptos principales*) analizando diferentes RCM, con el fin de detectar sinónimos, antónimos e inconsistencias. Así por ejemplo, el análisis de las definiciones *Alfabeto es un conjunto de símbolos*, *Lenguaje es un sistema de símbolos* y *Lenguaje es un conjunto de sistemas* permite la detección de posibles similitudes e inconsistencias entre los RCM de alfabeto y lenguaje. La métrica Similitud entre RCM (SRCM) se calcula como el valor medio de las similitudes de las definiciones (SD) de un conjunto de RCM.

3.4. Técnicas de representación del conocimiento

Como hemos indicado, antes de definir una nueva herramienta para evaluar y representar definiciones con sus referencias, realizamos un análisis de las herramientas existentes llegando a la conclusión de que ninguna de ellas era adecuada para nuestro propósito. En esta sección realizaremos un resumen de las principales técnicas analizadas, así como una comparativa con nuestra propuesta de RCM.

3.4.1. Relación de las técnicas analizadas

La mayoría de las propuestas sobre mapas de conceptos permiten representar los conceptos o ideas y sus relaciones pero existen diferencias entre ellas, en ocasiones bastante sutiles, lo que nos llevó a realizar un análisis más en profundidad para ver si alguna podría ser adecuada para nuestro propósito. De hecho, es bastante habitual confundirse entre ellas puesto que en ocasiones sus diagramas son muy similares (véase por ejemplo (McAleese, 1998; Milam et al., 2000), donde se usa el término *concept maps* en el contexto de los mapas de conocimiento o *knowledge maps*). A continuación se describen las principales opciones que fueron evaluadas:

- **Mind Maps**: se trata de una técnica propuesta por Buzan (1974) para la representación de ideas de forma esquemática e informal y que no era adecuada para nuestro propósito ya que al estar centrada en las *ideas* no permite mostrar con claridad la importancia de un concepto para comprender el significado de otro (Davies, 2010).
- **Knowledge Maps, Cognitive Maps and Argument Maps**: estas tres técnicas, propuestas respectivamente por O'Donnell et al. (2002), Eden (1988) y Twardy (2003) son similares a Mind Maps pero ofrecen la definición de tipos de relaciones entre ideas con el fin de mejorar el aprendizaje y la resolución de problemas. Al igual que con Mind Maps fueron descartadas por estar basadas en las relaciones entre ideas en lugar de conceptos. Además, *Knowledge Maps* y *Argument Maps* requieren asociar un tipo a las relaciones, lo que disminuye la flexibilidad a la hora de presentar las definiciones encontradas en la literatura.
- **Conceptual Graphs**: se trata de una interesante propuesta (Sowa, 2000) que permite visualizar las relaciones entre conceptos. Sin embargo, estas relaciones están restringidas por reglas basadas en la lógica y por lo tanto no pudimos encontrarlas adecuadas para representar definiciones en general.
- **Topic Maps**: es un estándar sobre la representación del conocimiento (Biezunski et al., 2002) que tiene su origen en el campo de la ontología y la semántica y que ofrece muchas posibilidades para almacenar de forma estructurada no sólo conceptos (denominados *topics*) y sus relaciones, sino también los recursos asociados con éstos, roles, nombres alternativos, etc. Sin embargo, aunque los *Topic Maps* pueden ser representados gráficamente, están más orientados hacia la representación textual mediante la utilización

del lenguaje XTM.

- **Annotation Ontologies**: también del campo de la ontología y la semántica, esta técnica (Ciccarese et al., 2010; Jovanovic and Gasevic, 2011) es una solución formal para enlazar metadatos basados en ontologías con otros recursos como documentos. Sin embargo, están más orientados hacia representaciones de la Web Semántica, especialmente la basada en OWL/RDF (Hayes et al., 2005). En nuestra opinión, sería una interesante solución complementaria para un problema cuyo objetivo fuese el inverso al nuestro, puesto que permite que a partir de un documento correspondiente a una referencia bibliográfica, permite enlazarle metadatos tales como los conceptos que aparecen en las definiciones incluidas en esa referencia.
- **Concept Maps**: finalmente, nuestra atención se centró en los *concept maps* conforme a la propuesta de Novak and Cañas (2008, 2006), quienes se basaron en los trabajos de Ausubel (1963) sobre la psicología del aprendizaje, quien afirmó que:

  > a meaningful learning only takes place when new concepts are connected to known concepts

  y efectivamente, como ya hemos indicado, los *concept maps* y por tanto los RCM, favorecen el pensamiento creativo al incentivar la creación de conocimiento a partir de la conexión entre conceptos. En la búsqueda hacia la solución de nuestro problema, pensamos que esta técnica era aplicable en lo referente a la comprensión de una definición en comparación con otras, visualizando qué conceptos comparten y cómo se relacionan. Otros interesantes aspectos de los *concept maps* que nos llevó a seleccionarlos como nuestro punto de partida fueron 1) su orientación hacia conceptos y relaciones, en lugar de una orientación a ideas, 2) su flexibilidad, su facilidad de uso y 3) el hecho de ser una de las soluciones más utilizadas en la actualidad, especialmente en el ámbito educativo.

  Por otro lado, aunque la flexibilidad de los *concept maps* podría ser considerada un inconveniente, (si se ve como un impedimento para su estudio formal), esta aparente limitación no ha impedido, tal como señalaba (Cañas and Carvalho, 2004), el análisis formal utilizando las herramientas y utilidades apropiadas para facilitar la conexión de *concept maps* con soluciones cercanas a la inteligencia artificial (ontologías, bases de datos léxicas, etc.). Del mismo modo, existen propuestas para conseguir la correspondencia en-

tre *concept maps* y otros estándares tales como los ya mencionados *topic maps* (Rovira, 2005) o *OWL/RDF* (Hayes et al., 2005). Esta complementariedad entre diversas soluciones a los mapas de conocimiento podría ser desarrollada aún más hacia la unificación de algunas de estas soluciones utilizando una herramienta común, tal como han sugerido algunos autores (Davies, 2010; Eppler, 2006).

3.4.2. Comparación entre *Concept maps* y RCM

Hemos visto que efectivamente *concept maps* son un buen punto de partida, pero no satisfacen todos nuestros requisitos. En *concept maps*, las proposiciones derivadas de los conceptos y sus relaciones están basadas únicamente en unidades semánticas formadas por conceptos enlazados por *linking phrases* (véase el apartado 3.2.3 sobre las contradicciones que podemos encontrar respecto a si esas unidades semánticas están restringidas a dos o a más conceptos).

Esta limitación es apuntada por AAhlberg (2004), quien indicó que:

> sometimes (...) the order in which propositions are read is important

y propuso una mejora de los *concept maps* para que incorporaran un número en cada enlace indicado el orden de lectura.

Incluso con esta mejora, los *concept maps* no nos permitirían representar diferentes proposiciones sobre el mismo conjunto de conceptos (véase el ejemplo de la figura 3.2), ya que sólo habría un número por enlace y por tanto cada terna (concepto - *linking phrase* - concepto) sólo podría formar parte de una proposición. RCM resuelve este problema de forma diferente al permitir a los *concept maps* la posibilidad de representar diferentes proposiciones usando *path labeling* y siendo el orden de lectura una consecuencia de la utilización de flechas dirigidas para enlazar conceptos.

Por otra parte, en los *concept maps*, la autoría de las proposiciones no es explícita (hecho que es especialmente patente cuando son creados colaborativamente) o en todo caso, se refiere a un único autor o autores sin poder identificar cuáles son sus aportaciones.

La utilización de *path labeling* introduce una nueva dimensión en los *concept maps* puesto que nos permite conocer el autor de una determinada proposición (definición en RCM). Como hemos descrito, esto permite realizar un interesante análisis centrado en los conceptos, las diferentes proposiciones sobre un *concep-*

to principal (definiciones) y las referencias bibliográficas asociadas (autoría), así como las relaciones entre ellas (que es un caso particular de *cross-links*).

Otro de los aspectos introducidos por RCM es la caracterización de la relevancia de los conceptos de una forma intuitiva, basada sobre criterios establecidos por el creador del RCM. Nuestra propuesta es que el creador del RCM analice la importancia de los conceptos presentes en una definición y asigne un *nivel* codificándolo mediante la intensidad de color (a mayor intensidad, mayor importancia). Esta sencilla solución facilita la identificación de los conceptos más relevantes para el autor independientemente de su localización en la jerarquía de conceptos del *concept map*.

No nos consta que exista una propuesta basada en *concept map* que combine definiciones y referencias bibliográficas. Existen soluciones independientes para ambos problemas: por ejemplo *Word Maps* (también conocidos como *Concept Definition Maps* (Schwartz and Raphael, 1985)), es una propuesta interesante para la enseñanza, especialmente a niños, del significado de las palabras utilizando diagramas. Por otro lado, *ConceptBib* (Le, 2004) propone la aplicación de *concept maps* a referencias y finalmente *CiteWiz* (Elmqvist and Tsigas, 2007) utiliza un *concept map* interactivo para mostrar las palabras clave y la autoría dentro de redes de citas científicas. Todas estas propuestas se centran por separado en definiciones, referencias y citaciones sin abordar la relación entre estos elementos.

3.4.3. Herramientas de diseño gráfico

Al haber basado RCM en *concept maps* revisamos aquellas herramientas como CMap Tools o VUE que facilitan la construcción de estos diagramas y ofrecen características adicionales como el enlace a recursos, compartición de *concept maps*, personalización de los estilos gráficos, etc. Sin embargo, no tenemos conocimiento de que exista una herramienta para *concept maps* que permita a los usuarios seleccionar un *path* y asignarle una etiqueta o asociarlo con un recurso (como por ejemplo un enlace que vincule al *path* con el documento referenciado).

La carencia de esta funcionalidad obliga a los creadores de RCM a 1) usar etiquetas de *linking phrases* o cajas de concepto (sin bordes) para contener las etiquetas de las referencias, realizando un uso de estos elementos gráficos para un propósito diferente al suyo, y 2) repetir el proceso de la asociación de referencias para cada frase de enlace a lo largo del *path*. Del mismo modo, la asociación de recursos a un path debería seguir el mismo proceso repetitivo.

Además aunque las herramientas para el diseño de *concept maps* generan las proposiciones (definiciones en RCM) incluidas en el diagrama en algún formato de texto, en el caso de los RCM esta extracción de definiciones no es satisfactoria por dos razones: 1) las definiciones extraídas pierden las referencias de las que proceden, y 2) en el caso, ya comentado, de las definiciones que están incluidas en otras definiciones, la herramienta no extrae las definiciones incluidas sino la definición con el *path* más largo.

Todas estas razones nos muestran que RCM propone características que mejoran los *concept maps* con nuevos elementos gráficos que no están presentes en las herramientas más utilizadas para el diseño de los *concept maps*.

3.4.4. Tablas comparativas

Para facilitar la comprensión y la comparación entre las diferentes opciones que hemos descrito y mostrar en detalle sus similitudes y diferencias con RCM, presentamos un resumen detallado en las tablas 3.4.4, 3.4.4 y 3.4.4 que complementa otros estudios comparativos como (Carnot et al., 2003; Davies, 2010; Elmqvist and Tsigas, 2007; Eppler, 2006; Gouveia, 2004; Le, 2004; Okada et al., 2008; Rovira, 2005). Para evitar ambigüedades, se han añadido las referencias que corresponden a cada propuesta, ya que existen algunas propuestas que comparten el mismo nombre. Nótese que se han recopilado las características conforme a las propuestas originales de los autores u ocasionalmente a mejoras que han sido generalmente aceptadas. Por ese motivo, podrían encontrarse variaciones o herramientas que extendiesen la funcionalidad de una determinada solución mediante la incorporación de características ofrecidas por otras soluciones.

Tabla 3.3: Resumen de técnicas de representación del conocimiento (1)

	Elementos	**Topología**	**Aplicaciones**	**Foco**	**Otros focos**	**Proposiciones**
RCM	Concepto principal Concept layering Path labelling References	Jerárquica	Análisis y comparación de definiciones y referencias bibliográficas	Main concept	No	Varios *paths* a través del mismo nodo
Concept Maps (Novak and Cañas, 2008)	Concepts Linking phrases	Jerárquica	Representación del conocimiento Mejora del aprendizaje	Focus question	Permitido	Grupos 2 o más conceptos

—Continúa en la página siguiente—

	Elementos	Topología	Aplicaciones	Foco	Otros focos	Proposiciones
Mind maps (Buzan, 1974)	Idea Asociation	Jerárquica	Brain-storming Retención de memoria	Central idea	No	Dentro de nodos
Conceptual graphs (Sowa, 2000)	Concepts Instances Conceptual relations Situations	Red	Razonamiento lógico	Proposition	Permitido	Única
Topic maps (Biezunski et al., 2002)	Topics Associations Resources Types Names	Red	Representación del conocimiento	No requerido	Permitido	No
Knowledge Maps (O'Donnell et al., 2002)	Ideas Links	Red	Mejora del aprendizaje	Central idea	Permitido	Dentro de nodos
Argument Maps (Twardy, 2003)	Claims Reasons Objections Rebuttals	Jerárquica	Aprender a razonar	Conclusion	No	Path a conclusion
Annotation ontology (Jovanovic and Gasevic, 2011)	Concepts Properties Relations	Red	Compartición metadatos	Resource	Permitido	No
Word maps (Schwartz and Raphael, 1985)	Main concept Category Properties Illustrations	Radial	Aprendizaje de vocabulario	Main concept	No	No
Cognitive maps (Eden, 1988)	Ideas Links	Red	Resolución de problemas	Focal point	Permitido	Dentro de nodos

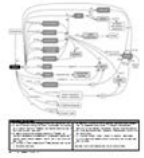
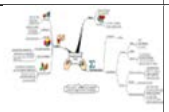
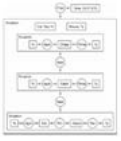
Tabla 3.4: Resumen de técnicas de representación del conocimiento (2)

	Orden lectura	Labeled links	Directional links	Cross links	Typed links	Typed nodes	Formatos gráficos	Estándar
RCM	Left to Right	Requerido	Unidireccional	Permitido	No	No	Colour-coded	No
Concept Map	Top to Bottom	Permitido	Requerido	Permitido	No	No	Permitido	No
Mind maps	Radial	No	Requerido	No	No	No	Permitido	No

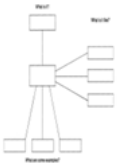
—Continúa en la página siguiente—

	Orden lectura	**Labeled links**	**Directional links**	**Cross links**	**Typed links**	**Typed nodes**	**Formatos gráficos**	**Estándar**
Conceptual graphs	Cualquiera	Requerido	Requerido	No	Requerido	Permitido	No	ISO/IEC 24707
Topic maps	Cualquiera	Requerido	Permitido	Permitido	Permitido	Sí	No	ISO/IEC 13250
Knowledge Maps	Cualquiera	Requerido	Requerido	No	Requerido	No	No	No
Argument Maps	Bottom to top	Permitido	Permitido	No	Requerido	No	Permitido	No
Annotation ontology	Radial	Requerido	Requerido	No	Requerido	Permitido	No	No
Word maps	Radial	No	No	No	No	No	No	No
Cognitive maps		Sólo "-"	Requerido	No	Permitido	No	Permitido	No

Tabla 3.5: Resumen de técnicas de representación del conocimiento (3)

	Formatos	**Enlace a recurso**	**Roles de nodo**	**Software**	**Formato gráfico**	**Muestra miniatura**	**Ref. Miniatura**
RCM	CXL, XTM, XCM, IVML	Permitido	No	En desarrollo	Requerido		Este documento
Concept Map	CXL, XTM, XCM, IVML	Permitido	No	CMAP Tools VUE	Requerido		(Novak and Cañas, 2008)
Mind maps	No	Permitido	No	FreeMind XMind iMindMap	Requerido		(Carnot et al., 2003)
Conceptual graphs	CGIF	No	No	CharGer Amine	Requerido		(Sowa, 2000)
Topic maps	XTM	Permitido	Permitido	Topic Map Designer Wandora DigiDocMap	Permitido		www.wandora.org

—Continúa en la página siguiente—

	Formatos	**Enlace a recurso**	**Roles de nodo**	**Software**	**Formato gráfico**	**Muestra miniatura**	**Ref. Miniatura**
Knowledge Maps	No	No	No	KP-Lab Tools	Requerido		(O'Donnell et al., 2002)
Argument Maps	AIF LKIF	No	No	Araucaria Carneades Rationale iLogos	Requerido		Wikimedia
Annotation ontology	RDF	Permitido	No	OBOEdit	Permitido		(Ciccarese et al., 2010)
Word maps	No	No	No	Genéricas	Requerido		(Schwartz and Raphael, 1985)
Cognitive maps	No	No	No	Decision Explorer	Requerido		(Eden, 1988)

Capítulo 4

Principales dificultades en el estudio de la modelización

En este capítulo describiremos las principales dificultades encontradas al estudiar la teoría de la modelización clasificadas en cuatro grupos: comprensión de los conceptos básicos, relaciones con la orientación a objetos, niveles de la modelización y los relacionados con la metamodelización. Esta revisión fue presentada y debatida en el Congreso MODELS 2010 (Rodriguez-Priego et al., 2010) donde despertó mucho interés entre los asistentes. El artículo fue enviado a OMG a instancias de los organizadores del Congreso, quienes nos transmitieron que el conocimiento de este trabajo podría ser de interés para la propia OMG dado que, según nos comunicaron, en aquellos momentos se estaba iniciando un proceso de revisión del estándar UML donde precisamente la complejidad y otras de las dificultades descritas en nuestro artículo estaban siendo revisadas.

Como ya hemos indicado, el origen de este trabajo fue la búsqueda de respuestas a algunas preguntas que surgieron en la definición de un metamodelo de seguridad para SOA. Así por ejemplo, encontramos que muchas de las propuestas relacionadas con nuestro campo de interés tenían que elegir entre un UML Profile o un metamodelo MOF, lo que nos llevó a plantearnos algunas preguntas tales como: ¿cuál de los dos mecanismos es el más adecuado para la definición de un metamodelo? ¿Cómo han de evaluarse esos mecanismos considerando que podrían dar lugar a resultados muy diferentes?. Y sobre todo, ¿cuál es el significado de los conceptos principales en los que están basados?

A partir de ese punto comenzamos un camino, aparentemente sin fin, revisando una gran cantidad de referencias sobre teoría de la modelización que nos

permitieran encontrar respuestas a estos interrogantes. Desafortunadamente pudimos comprobar, tal como hemos referido en el capítulo 3, que no había consenso en la literatura ni siquiera sobre un mínimo de conceptos básicos. Esto nos llevó a emplear mucho tiempo intentando comprender estos conceptos, sus inconsistencias, las ambigüedades de los estándares oficiales de OMG, etc.

El resultado de ese trabajo es el estudio que se describe a continuación y que puede ser abordado como una guía para comprender qué puntos de discusión impiden que la modelización pueda ser comprendida y tratada adecuadamente no sólo por los especialistas en este ámbito sino también por quiénes desean conocer sus fundamentos y aplicarla a la resolución de los problemas planteados en su ámbito de estudio.

4.1. Dificultades en la comprensión de los conceptos básicos de la modelización

En esta sección aplicaremos la técnica RCM, descrita en detalle en el capítulo 3, con el fin de facilitar la comprensión de las principales definiciones de modelización.

4.1.1. Modelo

El concepto central de la modelización es lógicamente el de modelo. Se trata de un concepto de uso cotidiano que puede tener muchos significados. En cierto modo, es como el concepto de objeto, que también es usado cotidianamente con diferentes definiciones. Sin embargo, mientras que el concepto de objeto aplicado a la computación ya tiene un significado razonablemente aceptado por la literatura, no ocurre así con el concepto de modelo.

Para abordar este problema, realizaremos el RCM de modelo siguiendo los pasos indicados en la sección 3.3.5 para la construcción de un RCM. realizando posteriormente un análisis basado en las métricas de los RCM descritas:

1. **Recopilación de las referencias**: tras la revisión de los artículos más relevantes sobre los conceptos de modelización pudimos comprobar que existía, y en el momento de escritura aún existe, una enorme confusión sobre el concepto de modelo (Börstler et al., 2012; Muller et al., 2009). De hecho en la definición de modelo aparecían conceptos muy diversos cuyas relaciones

entre ellos no eran fáciles de entender. La figura 4.1 recoge la lista de un total de 20 referencias seleccionadas.

[1] H. Stachowiak, *Allgemeine Modelltheorie*. Springer Wien, 1973.

[2] T. Asikainen and T. Männistö, "Nivel:a metamodelling language with a formal semantics}," *Software and Systems Modeling*, vol. 8, no. 4, pp. 521–549, Sep. 2009.

[3] C. Gonzalez-Perez and B. Henderson-Sellers, "Modelling software development methodologies: A conceptual foundation," *Journal of Systems and Software*, vol. 80, no. 11, pp. 1778–1796, Nov. 2007.

[4] E. Seidewitz, "What Models Mean," *IEEE Software*, vol. 20, no. 5, pp. 26–32, Sep. 2003.

[5] U. Aßmann, S. Zschaler, and G. Wagner, "Ontologies, Meta-models, and the Model-Driven Paradigm," *Ontologies for Software Engineering and Software Technology*, pp. 249–273, 2006.

[6] M. Pidd, *Tools for Thinking: Modelling in Management Science*, 3rd ed. Chichester, England: John Wiley & Sons, 2009.

[7] S. J. Mellor, K. Scott, A. Uhl, D. Weise, and R. M. Soley, *MDA distilled: principles of model-driven architecture*, vol. 88. New York: Addison-Wesley, 2004.

[8] J. Bézivin, "On the unification power of models," *Software and Systems Modeling*, vol. 4, no. 2, pp. 171–188, May 2005.

[9] M. Barbero, F. Jouault, and J. Bézivin, "Model Driven Management of Complex Systems:Implementing the Macroscope's Vision," in *{Engineering of Computer-Based Systems, IEEE International Conference on the}*, 2008, pp. 277–286.

[10] I. Ober and A. Prinz, "What do we need metamodels for?," in *4th Nordic Workshop on UML and Software Modelling*, 2006, pp. 8–28.

[11] T. Kühne, "Matters of (Meta-) Modeling," *Software and Systems Modeling*, vol. 5, no. 4, pp. 395–401, Dec. 2006.

[12] OMG, "MetaObject Facility (MOF) 1.4." Apr-2002.

[13] J. Bézivin, "Towards a precise definition of the OMG/MDA framework," in *Proceedings. 16th Annual International Conference on Automated Software Engineering, 2001.(ASE 2001)*, 2001, pp. 273–280.

[14] OMG, "MDA Guide Version 1.0.1." 12-Jun-2003.

[15] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained.The Model Driven Architecture: Practice and Promise*. Boston (USA): Addison-Wesley, 2003.

[16] A. Rensink, "Subjects, Models, Languages, Transformations," in *Dagstuhl Seminar Proceedings*, 2005, vol. 04101.

[17] D. M. Sánchez, J. M. Cavero, and E. Marcos, "On models and ontologies," in *1st Int. Workshop on Philosophical Foundations of Information Systems Engineering, PHISE 2005*, 2005, vol. 13.

[18] J.-M. Favre, "Foundations of meta-pyramids: Languages vs. metamodels Episode II: Story of thotus the baboon," *Language Engineering for Model-Driven Software Development*, vol. 4101, 2005.

[19] G. J. Klir, *Facets of Systems Science*. New York: Kluwer Academic/Plenum Publishers, 2001.

[20] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

Figura 4.1: Referencias del RCM de Modelo

2. **Identificación y reformulación de definiciones**: en este paso identificamos las definiciones contenidas en cada referencia y a partir de ellas las reformulamos extrayendo los conceptos y *linking-phrases* de sus respectivos *definientia*. La tabla de las figuras 4.2 y 4.3 muestran el resultado de esa tarea. Se han realizado capturas del texto de los artículos originales como muestra de las referencias para mayor claridad y se han resaltado los conceptos (mostrados con un recuadro de fondo gris) y *linking-phrases* (por un recuadro con fondo blanco) de las definiciones reformuladas. Las definiciones obtenidas se han numerado para su posterior identificación.

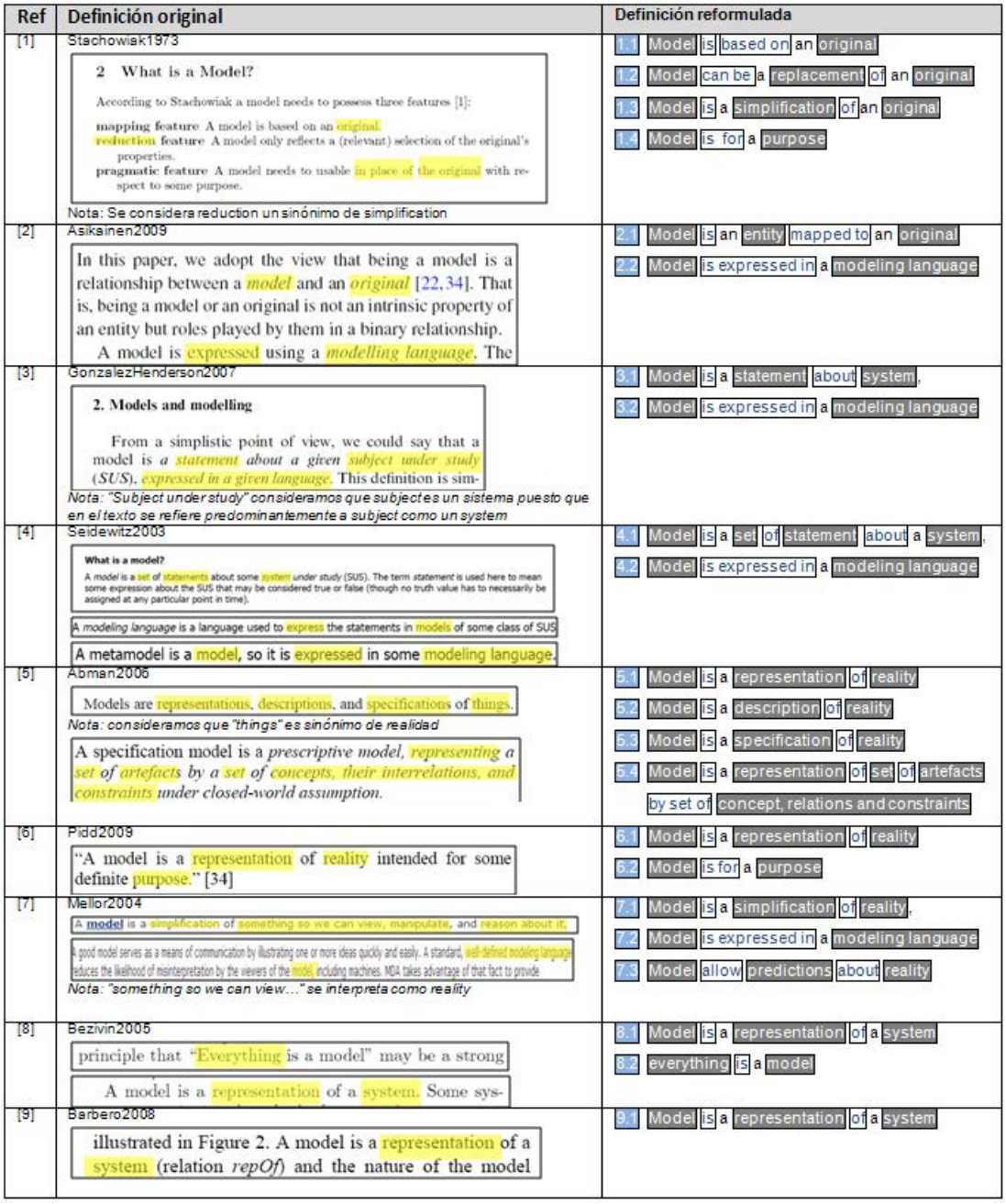


Figura 4.2: Identificación y reformulación de definiciones en el RCM de Modelo (1 de 2)

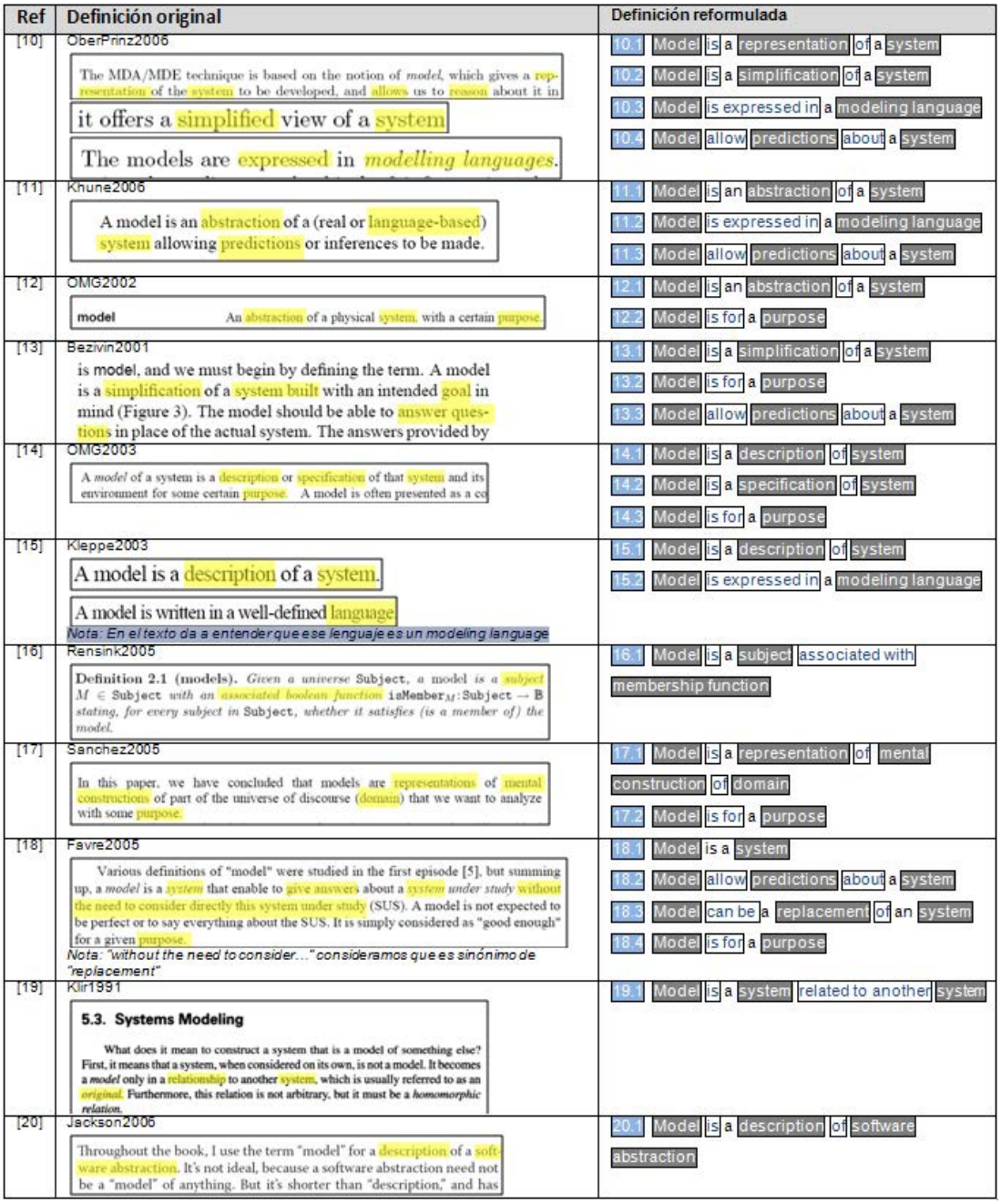


Figura 4.3: Identificación y reformulación de definiciones en el RCM de Modelo (2 de 2)

3. **Disposición de conceptos y *linking phrases***: tomando nuevamente como referencia la tabla de las figuras 4.2 y 4.3, se van disponiendo los conceptos en el diagrama enlazándolos con sus respectivas *linking-phrases*.

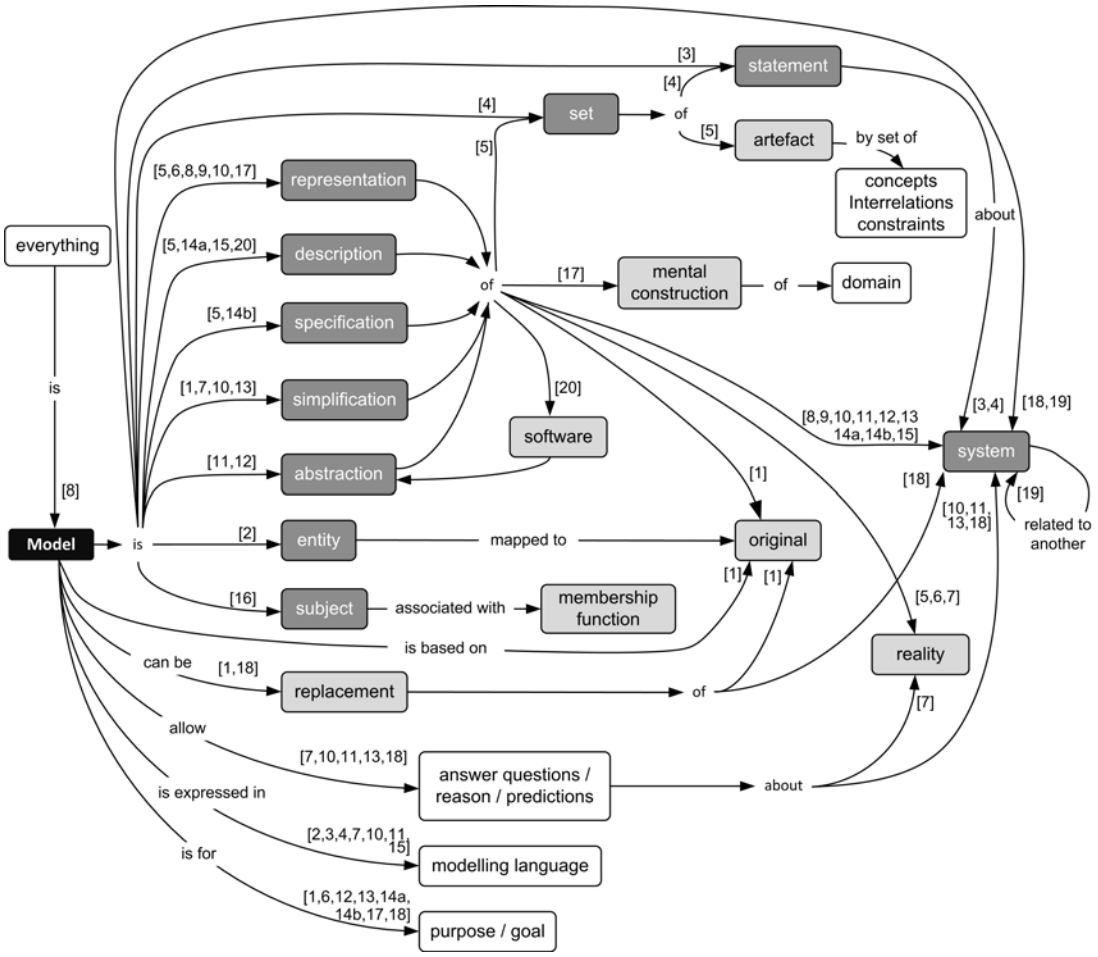


Figura 4.4: RCM Model

4. **Path labeling**: una vez colocados los conceptos y las *linking-phrases* se procede a realizar el *path labeling*. Para ello se toma nuevamente como referencia las tablas de las figuras 4.2 y 4.3 y se coloca la referencia correspondiente (en la primera columna) a cada definición (en la tercera columna).

5. **Concept layering**: el último paso es la elección y aplicación de un criterio para el *concept layering*. Decidimos que los *primary concepts* serán aquellos conceptos tras alguna forma del verbo *to be*, los *primary concepts* serán aquellos que les siguen y finalmente *tertiary concepts* el resto. Cuando algún concepto aparece en diferentes *niveles*, como por ejemplo el de *system*, se le asigna el de mayor importancia. Hay tres casos singulares: decidimos que las definiciones *Model is expressed in a modeling language*, *Model is for a purpose* y *Model allow predictions about a system*, se corresponden con el *nivel terciario* al considerar que se trata de definiciones explicativas (no del

tipo genus - differentia), que indican algunas características adicionales del *concepto principal*.

El resultado final puede observarse en la figura 4.4. Aunque mediante un examen visual pueden obtenerse ya algunas conclusiones, como su complejidad y la dispersión de conceptos, vamos a realizar un análisis más metódico mediante el cálculo de algunas de las métricas que fueron descritas en el apartado 3.3.8.

- **Métricas a nivel de *nivel***: la tabla de la figura 4.5 resume los datos generales del RCM y las métricas de dispersión a nivel de *niveles*.

RCM	D	R	N	C
Model	26	20	3	22

Layer	NCn	SFn	CSn	SSn
1	11	8	40%	28%
2	6	6	20%	20%
3	5	5	16%	16%

LEYENDA	
D	Número total de definiciones en el RCM
R	Número total de referencias en el RCM
N	Número total de layers en el RCM
C	Número total de conceptos en el RCM
NCn	Número de conceptos en el layer n
SFn	Número de semantic fields en el layer n
CSn	Concept scattering en el layer n
SSn	Semantic scattering en el layer n

Figura 4.5: Parámetros globales en RCM de Modelo

El RCM tiene un total de 26 definiciones tomadas de 20 referencias bibliográficas y un total de 22 conceptos. Se han considerado tres *niveles*. Hay prácticamente el doble de conceptos en el *nivel primario* respecto al resto de *niveles*. Por otro lado, se observa que la dispersión conceptual en el *nivel primario* es elevada (40 %) y va disminuyendo en los otros *niveles*. Respecto a los grupos semánticos, sólo aparecen en el *nivel primario* lo que se traduce en una menor dispersión semántica (28 %). En el resto la dispersión conceptual coincide con la semántica.

Concepto	GS (Grupo semántico)	References	Layer	rn(c)	DRCn(c)
Representation	G1.1	8,9,5,10,6,17	1	6	30%
Description	G1.1	5,14,15,20	1	4	20%
Specification	G1.2	5,14	1	2	10%
Simplification	G1.3	10,13,1,7	1	4	20%
Abstraction	G1.3	11,12	1	2	10%
Set	G1.4	4	1	1	5%
Replacement	G1.5	3,1,18	1	3	15%
Entity	G1.6	1,19,2	1	3	15%
Subject	G1.6	16	1	1	5%
System	G1.7	18,19,3,4,8,9,10,11,12,13,14,15	1	12	60%
Statement	G1.8	3,4	1	2	10%
original	G2.1	3,1,18,19,2	2	5	25%
reality	G2.2	5,6	2	2	10%
Member function	G2.3	16	2	1	5%
mental construction	G2.4	17	2	1	5%
artefact	G2.5	5	2	1	5%
software	G2.6	20	2	1	5%
Modeling-language	G3.1	2,3,4,7,19,11,15	3	7	35%
Purpose	G3.2	1,6,12,13,14,17,18	3	7	35%
Predictions	G3.3	7,10,11,13,18	3	5	25%
concepts, relations and constraints	G3.4	5	3	1	5%
domain	G3.5	17	3	1	5%

LEYENDA

rn(c)	Número de referencias que llegan al concepto c en el layer n
DRCn(c)	Densidad de referencias del concepto c en el layer n

Figura 4.6: Lista de conceptos en RCM de Modelo

Estos datos reflejan que las discusiones en la literatura se han centrado sobre todo en el primer *nivel* que según nuestro criterio, utilizado para la creación del RCM, se corresponde con el *genus* de las definiciones. En efecto, la *dispersión conceptual* del primer *nivel* es el doble que el del segundo y éste más del doble que el del tercero. No hay, por tanto, consenso sobre qué es en primer término un modelo. Por otro lado, si observamos la dispersión desde el punto de vista de los grupos semánticos, observamos que la distancia entre la *dispersión semántica* del primer y segundo *nivel* es menor (un 28 % frente a un 40 % en *dispersión conceptual*) mientras que no hay ningún cambio respecto a los siguientes *niveles*. Esto indica que algunos autores han utilizado términos con significados similares, pero no lo suficiente como para concluir que exista un acuerdo sobre cuál es el *genus* de modelo. Más bien al contrario, los matices que introduce la utilización de términos parecidos puede confundir aún más sobre cuál es el significado correcto de modelo.

- **Métricas a nivel de Concepto y Grupo semántico**: la figura 4.6 muestra los conceptos del RCM clasificados por niveles. La última columna contiene la densidad de referencias para cada concepto. El concepto con mayor densidad es *system* con un 60 %, seguido de dos conceptos terciarios (*mo-*

deling language y *purpose*) con el 35 %. El siguiente es *representation* con un 30 %. El resto de conceptos primarios tiene una densidad media inferior al 15 %. El concepto secundario de mayor densidad es *original* con un 25 %, pero el resto de conceptos de ese nivel tiene una densidad media inferior al 10 %.

Grupo	Conceptos	rn(sf)	DRSFn(sf)
G1.1	Representation Description	10	50%
G1.2	Specification	2	10%
G1.3	Simplification Abstraction	6	30%
G1.4	Set	1	5%
G1.5	Replacement	3	15%
G1.6	Subject Entity	4	20%
G1.7	System	12	60%
G1.8	Statement	2	10%
G2.1	Original	5	25%
G2.2	Reality	2	10%
G2.3	Member function	1	5%
G2.4	Mental construction	1	5%
G2.5	Artefact	1	5%
G2.6	Software	1	5%
G3.1	Modeling language	7	35%
G3.2	Purpose	7	35%
G3.3	Predictions	5	25%
G3.4	concepts, relations and	1	5%
G3.5	domain	1	5%

LEYENDA

rn(sf)	Número de referencias que llegan al semantic field sf en el layer n
DRSFn(sf)	Densidad de referencias del semantic field sf en el layer n

Figura 4.7: Lista de Grupos semánticos en RCM de Modelo

Estos datos indican que no sólo hay una elevada dispersión conceptual en las definiciones sino que también hay poca coincidencia en la literatura sobre cuáles son los conceptos primarios y secundarios más importantes. Por el contrario, sí que hay mayor coincidencia en las definiciones explicativas del *nivel terciario*. En la figura 4.7 se presentan los mismos cálculos de densidad pero para los grupos semánticos. Centraremos nuestra atención en el *nivel primario* ya que, como hemos visto no existen grupos diferentes en el resto de *niveles*. En este caso, el grupo semántico de *system* sigue siendo el de mayor densidad, pero la densidad media del resto de grupos semánticos de ese nivel es prácticamente el doble de la densidad conceptual correspondiente. En efecto, al considerar los grupos semánticos, el 50 % de las referencias consideran que el grupo semántico *(Representation, Description)* es un *genus* en la definición de modelo.

Se concluye por tanto que, aún siendo bajo el consenso, la literatura consigue un grado mayor de acuerdo respecto al significado de los conceptos en el *nivel primario*.

- **Métricas a nivel de definiciones**: la figura 4.8 muestra la relación de definiciones numeradas de D01 a D26 indicando la correspondencia con sus referencias según su numeración (ver la tabla de las figuras 4.2 y 4.3). Esta figura también muestra las repeticiones de las definiciones en la literatura.

D Id	Definición	Ref
D01	Model allows Predictions about Reality	7.3
D02	Model allows Predictions about System	10.4, 11.3, 13.3, 18.2
D03	Model can be Replacement of Original	1.2
D04	Model can be Replacement of System	18.3
D05	Model is a Abstraction of System	11.1, 12.1
D06	Model is a Description of Reality	5.2
D07	Model is a Description of Software Abstraction	20.1
D08	Model is a Description of System	14.1, 15.1
D09	Model is a Representation of Mental-construction of Domain	17.1
D10	Model is a Representation of Reality	5.1, 6.1
D11	Model is a Representation of Set of Artefact by Set of Concept-Relations-Constraints	5.4
D12	Model is a Representation of System	8.1, 9.1, 10.1
D13	Model is a Set of Statement about a System	4.1
D14	Model is a Specification of Reality	5.3
D15	Model is a Specification of System	14.2
D16	Model is a Statement about a System	3.1
D17	Model is a Subject with associated Member-function	16.1
D18	Model is a System	18.1
D19	Model is a System related to another System	19.1
D20	Model is Entity mapped to Original	2.1
D21	Model is expressed in Modeling-language	2.2, 3.2, 4.2, 7.2 ,10.3, 11.2, 15.2
D22	Model is for a Purpose	1.4, 6.2, 12.2, 13.2, 14.3, 17.2, 18.4
D23	Model is Simplification of Original	1.3
D24	Model is Simplification of Reality	7.1
D25	Model is Simplification of System	10.2, 13.1
D26	Model is based on Original	1.1

Figura 4.8: Definiciones del RCM de Modelo

- **Calidad de las definiciones:**
  - **Esencialidad:** tomando como referencia la lista de definiciones anterior, nos centraremos en el cálculo de la *Esencialidad de las definiciones* según la fórmula de la Definición 3.9. Las columnas 2 y 3 de la tabla de la figura 4.9 muestran la densidad referencial de cada definición y su nivel. En las columnas siguientes se muestra el valor de la *Relevancia del concepto* CR, sólo para aquellos conceptos que aparecen en cada definición. Finalmente la última fila presenta el valor de la *Esencialidad de la definición* DE, obtenido mediante la suma de los respectivos CR de los conceptos de cada definición.

Concepto	DRCn(c)	n	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26
Representation	30	1									15	15	15	15														
Description	20	1						10	10	10							10											
Specification	10	1														5												
Simplification	20	1																							10	10	10	
Abstraction	10	1					5		5																			
Set	5	1											3		3													
Replacement	15	1			8	8																						
Entity	15	1																				8						
Subject	5	1																	3									
System	60	1		30		30	30			30				30	30		30	30		30	30						30	
Statement	10	1													5			5										
original	25	2			8																	8			8			8
reality	10	2	3					3				3				3										3		
Member function	5	2																	2									
mental construction	5	2									2																	
artefact	5	2											2															
software	5	2							2																			
Modeling-language	35	3																					6					
Purpose	35	3																						6				
Predictions	25	3	4	4																								
concepts, relations and domain	5	3											1															
	5	3									1																	
	DE(d)		8	34	16	38	35	13	17	40	18	18	20	45	38	8	40	35	4	30	30	16	6	6	18	13	40	8

LEYENDA

DRCn(c)	Densidad de referencias del concepto c en el layer n
DE(d)	Definition essentiality de la definición d

Figura 4.9: Esencialidad de las definiciones en RCM de Modelo

Los datos reflejan que las definiciones más relevantes son D12 (*Model is a Representation of System*), D08 (*Model is a Description of System*), D15 (*Model is a Specification of System*) y D25 (*Model is Simplification of System*) con una esencialidad mayor o igual a 40.

- **Circularidad**: para el cálculo de esta métrica se necesitaría el RCM de los conceptos que aparecen en el RCM de modelo. Así por ejemplo si del RCM de original se obtuviera que *original is a model* podríamos medir el número de referencias circulares sobre el total de definiciones.
- **Genericidad**: para el cálculo de esta métrica también sería necesario contar con los RCM de otros *concepto principal* para así poder detectar, por ejemplo, si existen definiciones que coincidan en genericidad. En un primer examen se observa que algunas definiciones podrían ser demasiado generales (por ejemplo, D18 *model*

is a system).

- **Claridad**: todas las definiciones están expresadas con un lenguaje claro y con conceptos que tienen un significado concreto.
- **Asertividad**: todas las definiciones están expresadas de manera afirmativa.

- **Similitudes entre definiciones:** veamos a continuación el cálculo de las similitudes entre definiciones. La tabla de la figura 4.10 es una matriz que compara cada definición con el resto indicándose en cada casilla el valor de la similitud entre dos definiciones según la fórmula de la sección 3.3.8. La última fila presenta el valor de la similitud calculando el valor medio de las similitudes obtenidas por definición.

D Id	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26
D01	X	50%	0%	0%	0%	50%	0%	0%	0%	50%	0%	0%	0%	50%	0%	0%	0%	0%	0%	0%	0%	0%	0%	50%	0%	0%
D02	50%	X	0%	50%	50%	0%	0%	50%	0%	0%	0%	50%	33%	0%	50%	50%	0%	50%	50%	0%	0%	0%	0%	0%	50%	0%
D03	0%	0%	X	50%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	50%	0%	0%	50%	0%	0%	50%
D04	0%	50%	50%	X	50%	0%	0%	50%	0%	0%	0%	50%	50%	0%	50%	50%	0%	50%	50%	0%	0%	0%	0%	0%	50%	0%
D05	0%	50%	0%	50%	X	0%	33%	50%	0%	0%	0%	50%	50%	0%	50%	50%	0%	50%	50%	0%	0%	0%	0%	0%	50%	0%
D06	50%	0%	0%	0%	0%	X	33%	50%	0%	50%	0%	0%	0%	50%	0%	0%	0%	0%	0%	0%	0%	0%	0%	50%	0%	0%
D07	0%	0%	0%	0%	33%	33%	X	33%	0%	0%	0%	50%	33%	0%	50%	50%	0%	50%	50%	0%	0%	0%	0%	0%	50%	0%
D08	0%	50%	0%	50%	50%	50%	33%	X	0%	0%	0%	50%	33%	0%	50%	50%	0%	50%	50%	0%	0%	0%	0%	0%	50%	0%
D09	0%	0%	0%	0%	0%	0%	0%	0%	X	33%	20%	33%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
D10	50%	0%	0%	0%	0%	50%	0%	0%	33%	X	20%	50%	0%	50%	0%	0%	0%	0%	0%	0%	0%	0%	0%	50%	0%	0%
D11	0%	0%	0%	0%	0%	0%	0%	0%	20%	20%	X	20%	20%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
D12	0%	50%	0%	50%	50%	0%	50%	50%	33%	50%	20%	X	33%	0%	50%	50%	0%	50%	50%	0%	0%	0%	0%	0%	50%	0%
D13	0%	33%	0%	50%	50%	0%	33%	33%	0%	0%	20%	33%	X	0%	33%	67%	0%	33%	33%	0%	0%	0%	0%	0%	33%	0%
D14	50%	0%	0%	0%	0%	50%	0%	0%	0%	50%	0%	0%	0%	X	50%	0%	0%	0%	0%	0%	0%	0%	0%	50%	0%	0%
D15	0%	50%	0%	50%	50%	0%	50%	50%	0%	0%	0%	50%	33%	50%	X	50%	0%	50%	50%	0%	0%	0%	0%	0%	50%	0%
D16	0%	50%	0%	50%	50%	0%	50%	50%	0%	0%	0%	50%	67%	0%	50%	X	0%	50%	50%	0%	0%	0%	0%	0%	50%	0%
D17	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	X	0%	0%	0%	0%	0%	0%	0%	0%	0%
D18	0%	50%	0%	50%	50%	0%	50%	50%	0%	0%	0%	50%	33%	0%	50%	50%	0%	X	50%	0%	0%	0%	0%	0%	50%	0%
D19	0%	50%	0%	50%	50%	0%	50%	50%	0%	0%	0%	50%	33%	0%	50%	50%	0%	50%	X	0%	0%	0%	0%	0%	50%	0%
D20	0%	0%	50%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	X	0%	0%	50%	0%	0%	50%
D21	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	X	0%	0%	0%	0%	0%
D22	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	X	0%	0%	0%	0%
D23	0%	0%	50%	0%	0%	0%	0%	0%	0%	50%	0%	0%	0%	0%	0%	0%	0%	0%	0%	50%	0%	0%	X	50%	50%	50%
D24	50%	0%	0%	0%	0%	50%	0%	0%	0%	0%	0%	0%	0%	50%	0%	0%	0%	0%	0%	0%	0%	0%	50%	X	50%	0%
D25	0%	50%	0%	50%	50%	0%	50%	50%	0%	0%	0%	50%	33%	0%	50%	50%	0%	50%	50%	0%	0%	0%	50%	50%	X	0%
D26	0%	0%	50%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	50%	0%	0%	50%	0%	0%	X
SDd	10%	21%	8%	22%	21%	11%	17%	23%	3%	12%	3%	25%	18%	10%	23%	23%	0%	21%	21%	6%	0%	0%	12%	10%	25%	6%

Figura 4.10: Similitudes de definiciones en RCM de Modelo

Se observa que las cuatro primeras definiciones con más similitudes DE coinciden con las más relevantes (D12, D25, D15, D08), lo cual puede interpretarse como que además de ser relevantes ofrecen variaciones con matices diferentes. En el otro extremo, la única definición del tipo *genus y differentia* que no tiene nada en común con el resto es D17 (*Model is a subject with associated membership function*).

- **Dependencias entre definiciones**: se ha observado una dependencia entre definiciones. Consiste en que todas las referencias que usan el

concepto *specification* también incluyen el concepto *description* pero no viceversa. Esto indica que los autores consideran que ambos aspectos (*description* y *specification*) están ligados por algún tipo de relación. Efectivamente en este caso están indicando que hay dos propuestas básicas a la definición de modelo que serían incompletas si sólo se considera una de ellas.

- **Contradicciones e inconsistencias**: se observa una inconsistencia en las definiciones. Se trata de las definiciones D13 (*Model is a set of statements about a system*) y D18 (*Model is a system*). Si se considera la definición clásica de sistema (Klir, 2001) como una entidad con elementos relacionados podemos deducir que un modelo no puede ser a la vez un conjunto (set) y un sistema, ya que en un conjunto los elementos no están necesariamente interrelacionados mientras que en un sistema sí.

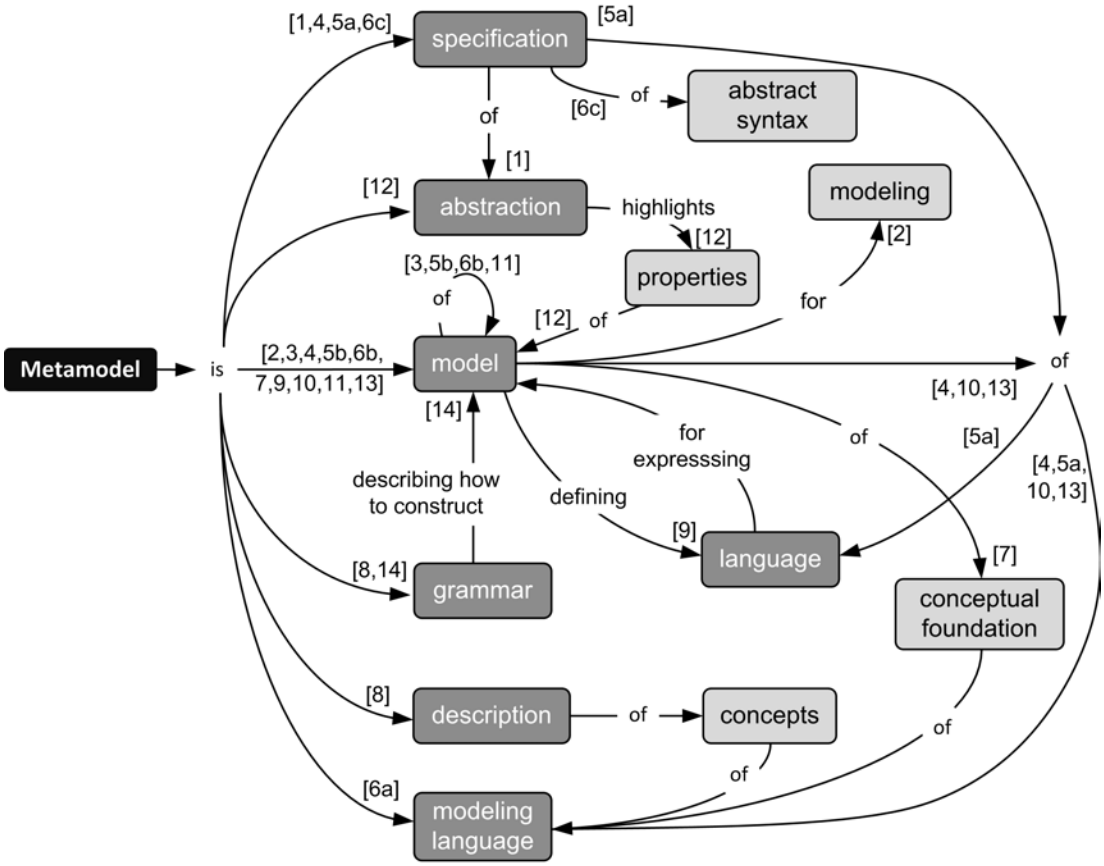
- **Métrica a nivel de RCM**: para obtener la métrica *Similitudes entre RCM* se necesitaría una mayor cantidad de RCM con el fin de comprobar si existen sinonimias, antonimias etc. entre ellos.

A modo de resumen, podemos concluir que el RCM de Modelo es relativamente complejo, existe dispersión conceptual y en menor medida semántica, tiene un pequeño número de definiciones esenciales con diferentes variaciones pero sobre las que no existe un gran consenso y se han detectado dependencias e inconsistencias entre ellas.

A la vista de este análisis podríamos plantearnos algunos interrogantes que no son sencillos de responder: ¿Cuál es la definición correcta? ¿Son todas correctas?. Por el momento, y tras ya largos años de discusiones sobre estos conceptos, lo único claro es que se están creando y usando modelos sin conocerse realmente su significado (Muller et al., 2009).

4.1.2. Metamodelo

En el apartado anterior hemos descrito en detalle el método de obtención de un RCM aplicado al de Modelo, así como el análisis del mismo a través de las métricas descritas en el capítulo 3. En este apartado y el siguiente, presentamos de una forma más resumida, los resultados de aplicar la misma técnica a otros conceptos relevantes de la modelización.



[1] J. Bézivin, "On the unification power of models," *Software and Systems Modeling*, vol. 4, no. 2, pp. 171–188, May 2005.
[2] OMG, "MetaObject Facility (MOF) 2.0 Core specification." Jan-2006.
[3] T. Kühne, "Matters of (Meta-) Modeling," *Software and Systems Modeling*, vol. 5, no. 4, pp. 395–401, Dec. 2006.
[4] E. Seidewitz, "What Models Mean," *IEEE Software*, vol. 20, no. 5, pp. 26–32, Sep. 2003.
[5] U. Aßmann, S. Zschaler, and G. Wagner, "Ontologies, Meta-models, and the Model-Driven Paradigm," *Ontologies for Software Engineering and Software Technology*, pp. 249–273, 2006.
[6] C. Gonzalez-Perez and B. Henderson-Sellers, "Modelling software development methodologies: A conceptual foundation," *Journal of Systems and Software*, vol. 80, no. 11, pp. 1778–1796, Nov. 2007.
[7] I. Kurtev, "Metamodels: Definitions of Structures or Ontological Commitments," in *Workshop on TOWERS of models. Collocated with TOOLS Europe 2007*, 2007, vol. 2007.
[8] I. Ober and A. Prinz, "What do we need metamodels for?," in *4th Nordic Workshop on UML and Software Modelling*, 2006, pp. 8–28.
[9] OMG, "MetaObject Facility (MOF) 1.4." Apr-2002.
[10] J.-M. Favre, "Foundations of meta-pyramids: Languages vs. metamodels Episode II: Story of thotus the baboon," *Language Engineering for Model-Driven Software Development*, vol. 4101, 2005.
[11] OMG, "MDA Guide Version 1.0.1." 12-Jun-2003.
[12] A. Pierantonio and S. Zacchiroli, "Metamodel for Describing System Structure and State," 2009. [Online]. Available: http://upsilon.cc/~zack/research/publications/mancoosi-d2.1.pdf.
[13] S. J. Mellor, K. Scott, A. Uhl, D. Weise, and R. M. Soley, *MDA distilled: principles of model-driven architecture*, vol. 88. New York: Addison-Wesley, 2004.
[14] G. H. Joyce, *Principles of logic*. London: Longmans, Green and Co., 1916.

Figura 4.11: RCM Metamodel

El segundo término que más aparece en la literatura sobre los conceptos de modelización es el de metamodelo. La figura 4.11 muestra el RCM de metamodelo construido a partir de 14 referencias y que está formado por 7 conceptos primarios y 5 secundarios. Es interesante comparar este RCM con el de Modelo puesto que

hay un número elevado de referencias que lo definen como un tipo especial de modelo. Efectivamente, analizando los conceptos del *nivel primario* se observa que todos ellos menos *grammar* son también conceptos primarios en el RCM de modelo. Esto confirma que la mayoría de los autores creen que *modelo* es el *genus* de *metamodelo*.

Podemos por tanto identificar dos grupos semánticos en el *nivel primario*:

1. El formado por los conceptos *model*, *description*, *specification*, *abstraction*, *representation*. Los autores de este grupo definen *metamodelo* como un caso particular de modelo en el que el *nivel secundario*, es decir el sistema que se modeliza, es a su vez un *modelo*, la *modelización* en sí misma o algún concepto relacionado con el lenguaje (*abstract syntax*, *modeling language*).

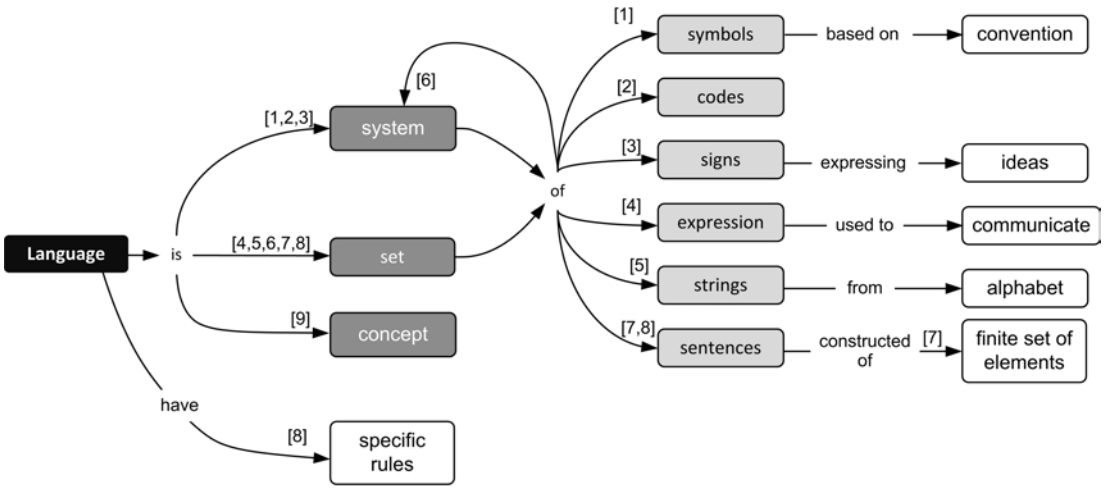
2. El relacionado con el lenguaje (*grammar* y *modeling language*) que está respaldado por tres autores en el *nivel primario*.

Podemos preguntarnos nuevamente ¿cuál es la definición correcta? ¿Son todas ellas correctas?

4.1.3. Lenguaje y Lenguaje de modelización

Para el estudio del RCM de *lenguaje de modelización* hemos creído conveniente realizar y analizar brevemente el RCM de Lenguaje construido a partir del estudio de 9 referencias. Como puede observarse en la figura 4.12 la mayoría de los autores piensan que el *genus* de *lenguaje* lo forman principalmente dos conceptos primarios (*set* y *system*), salvo un autor que cree que un lenguaje es un concepto (una abstracción). Obsérvese que aquí también se produce una inconsistencia, pues como ya indicamos a propósito de las métricas sobre inconsistencias en las definiciones (ver apartado 4.1.1), afirmar que algo es un conjunto y un sistema es contradictorio ya que en un sistema los elementos están relacionados mientras que en un conjunto no.

En el *nivel secundario* aparecen un mayor número de conceptos, todos ellos similares: *symbols*, *codes*, *sentences*, *strings*, *expression*, *signs*. Llama la atención que salvo una referencia, el resto utiliza un concepto diferente para este *nivel*. Finalmente en el *nivel terciario* hay un número similar de conceptos y son también diferentes para cada referencia.



[1] R. Robins, "General linguistics: An introductory survey," 1971.
[2] A. R. Luria, *Language and Cognition*. John Wiley & Sons Inc, 1979.
[3] F. De Saussure, "Course in General Linguistics," *London: Fontana*, vol. 240, 1974.
[4] D. Harel and B. Rumpe, "Modeling Languages: Syntax, Semantics and All That Stuff, Part I:The Basic Stuff," Weizmann Science Press of Israel, MCS00-16, 2000.
[5] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*, 3rd ed. Addison Wesley, 2006.
[6] J.-M. Favre, "Foundations of meta-pyramids: Languages vs. metamodels Episode II: Story of thotus the baboon," *Language Engineering for Model-Driven Software Development*, vol. 4101, 2005.
[7] N. Chomsky, "Syntactic structures. 1957," *Mouton, The Hague*, vol. 2, 1957.
[8] M. A. Nowak, N. L. Komarova, and P. Niyogi, "Computational and evolutionary aspects of language," *Nature*, vol. 417, no. 6889, pp. 611–617, Jun. 2002.
[9] T. Kühne, "Matters of (Meta-) Modeling," *Software and Systems Modeling*, vol. 5, no. 4, pp. 369–385, Dec. 2006.

Figura 4.12: RCM Language

Pasemos ahora al RCM de *lenguaje de modelización* representado en la figura 4.13. Está realizado a partir de 8 referencias y está compuesto por 5 conceptos en el *nivel primario*, 6 en el *nivel secundario* y 1 en el *nivel terciario*.

Puede observarse que no hay acuerdo respecto a cuál es el *genus* de *lenguaje de modelización*. Sin embargo, la mitad de los autores convienen en que un lenguaje de modelización es un tipo especial de lenguaje.

Tampoco hay acuerdo respecto al *nivel secundario*, donde el número de conceptos es ligeramente mayor. No obstante, la mayoría de los autores apuntan a que el concepto *model* es la *differentia* de *lenguaje de modelización*. Por otro lado, se observa que en el *nivel primario* no existen grupos semánticos, pero en el *nivel secundario model unit kinds* y *metamodel definition* podrían estar en un mismo *campo semántico* dado que ambos conceptos ser refieren al conjunto de elementos que definen un metamodelo.

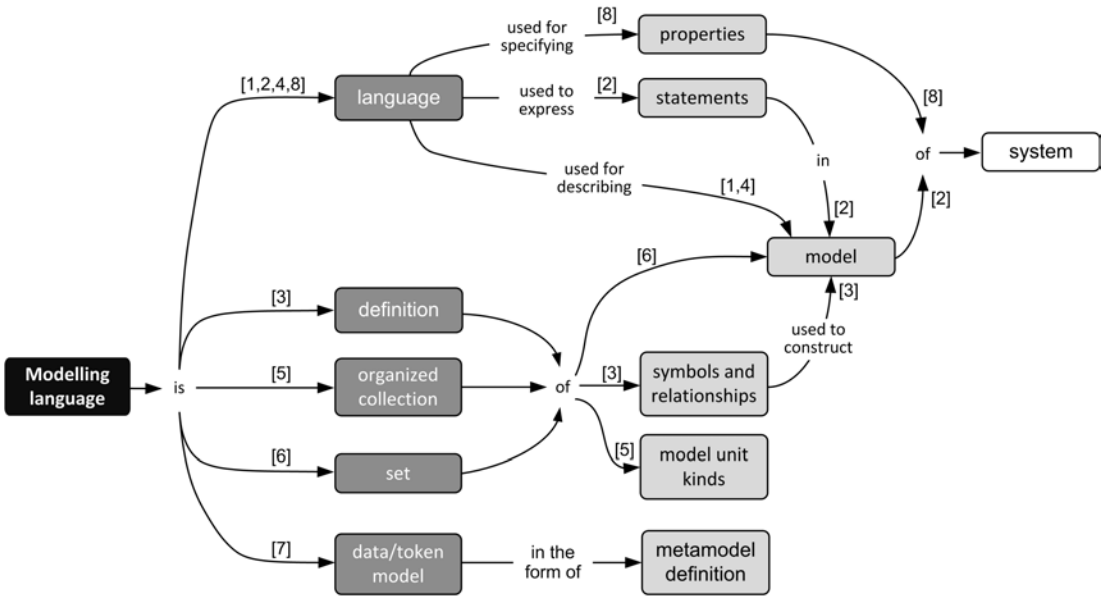


Figura 4.13: RCM Modeling Language

Finalmente, si los comparamos con el RCM de lenguaje, podemos ver que aunque *set* es un *genus* de lenguaje, no podemos incluir la proposición *set of models* en el mismo grupo semántico que *language* dado que el concepto *model* no se encuentra en el *nivel secundario* de *language*.

Una vez más, el análisis del RCM nos revela que no existe consenso sobre el significado del *concepto principal* y volvemos a preguntarnos cual es la definición correcta y si todas ellas son correctas.

4.2. Dificultades sobre las relaciones entre modelización y orientación a objetos

Hemos podido comprobar que el análisis de los RCM de los conceptos de modelización conduce al planteamiento de nuevas dudas y a la constatación de que existe cierta confusión por la falta de consenso entre los autores. Además de este

problema, nos encontramos con otro gran obstáculo, que es el de cómo enfocar la definición de los modelos que queremos realizar y qué lenguaje utilizar para expresarlos.

La literatura responde categóricamente a esa cuestión: el enfoque predominante es el del *Object Management Group* (OMG) creador del lenguaje de modelización más utilizado: UML. Era de esperar que la utilización de este enfoque ayudase incluso a clarificar los conceptos de modelización. Sin embargo, la revisión de una extensa literatura sobre el tema[1] condujo a un arduo camino con diferentes ramificaciones que puso de manifiesto una serie de dificultades en la aplicación del enfoque OMG y de la modelización en general.

En esta sección y siguientes se describen las principales dificultades encontradas clasificadas en tres grupos: 1) Relaciones entre modelización y orientación a objetos, 2) Niveles de la modelización y 3) Metamodelización.

4.2.1. El enfoque de OMG orientado a objetos sobre la modelización

A la vista de las definiciones sobre modelización indicadas en 4.1 sería de esperar que el enfoque más utilizado hiciera uso de conceptos como modelo, sistema, descripción, especificación, abstracción, etc. Sin embargo, OMG (que es una organización centrada en la orientación a objetos) enfoca la modelización como una aplicación de la orientación a objetos y por tanto utiliza conceptos como clase, instancia y herencia en todas sus especificaciones de modelización (Favre, 2004a).

En realidad se trata de algo más que la utilización de unos u otros conceptos, pues tal como indica Bézivin (2005) en su clásico artículo:

> There is presently an important paradigm shift in the field of software engineering that may have important consequences on the way information systems are built and maintained (...) The central idea of object composition is progressively being replaced by the notion of model transformation.

El problema es que al predominar el enfoque OMG orientado a objetos de la modelización, en lugar de un cambio de paradigma lo que se produce es una

[1] La lista completa de la literatura analizada para la elaboración de este capítulo fue de 215 referencias. Por razones de espacio, sólo 93 de esa lista están incluidas en la bibliografía de la presente memoria. No obstante, a raíz de la publicación de (Rodriguez-Priego et al., 2010) en el Congreso Models 2010, la lista completa fue compartida a través de un grupo del sistema de gestión de referencias Mendeley y puede ser consultada libremente en http://www.mendeley.com/groups/516691/model-theory/

combinación de ambos paradigmas con un resultado muy confuso. Pasamos del *todo es un objeto* a *todo es un modelo*... pero sin dejar de ser un objeto. Favre (2004b) lo expresa con absoluta claridad:

> It happens that object-oriented technologies are used to *implement* MDA standards, but this is just the result of a particular choice (...) Simply put Model Driven Engineering is not about Object Orientation.

Sin embargo, la mayoría de los autores que abordan de alguna forma los fundamentos de la modelización terminan utilizando el enfoque OMG, sin proponer una separación clara entre ambos paradigmas para facilitar una transición real de un paradigma a otro.

En nuestra opinión uno de los factores que ha facilitado la confusión entre modelización y la orientación a objetos es que la implementación de modelos mediante clases produce un homomorfismo entre los conceptos de modelización y los de orientación a objetos (Gonzalez-Perez and Henderson-Sellers, 2007). En efecto, las relaciones entre un modelo y la entidad modelizada son similares (homomorfos) a las de una clase y su instancia. Esto puede ser una ventaja inicial para quienes ya conocen la orientación a objetos, pero se torna un inconveniente cuando se mezclan conceptos que en el fondo no tienen nada que ver. Así por ejemplo, en (OMG, 2011c) se define *model* como un *package*, definido a su vez como un contenedor de *element* (más concretamente de un tipo especial de elementos denominados *packageable elements*). Por otro lado se define *element* como *a constituent of a model* y añade que *element is an abstract metaclass with no superclass. It is used as the common superclass for all metaclasses in the infrastructure library*. Es decir, para OMG un modelo es una agrupación de clases que a su vez son subclases de una superclase común denominada *element*. Por otro lado, *package* es una subclase de *packageableElement* que a su vez es una subclase de *element*. Por tanto, un modelo, según OMG, también es una *class*.

Es difícil encontrar una correlación entre las diferentes definiciones de modelo indicadas en la sección 4.1.1 y esta definición proporcionada por OMG, que mezcla conceptos de modelización con conceptos de orientación a objetos, más aún si se analizan las implicaciones de considerar que un modelo es un *package*, un *element* y una *class* al mismo tiempo. Así por ejemplo, un *package* es también un *namespace* y una *class* tiene operaciones y estado. Intuitivamente no se entiende que un modelo tenga operaciones y estado o que sea un espacio de nombres, y sin embargo no existan referencias sobre teoría de la modelización que mencionen

que un modelo tenga dichas características.

4.2.2. Sobreutilización del concepto de Clase

Aunque el enfoque de OMG fuese el más adecuado para la modelización encontraríamos muchas dificultades para su comprensión y aplicación. Los principales documentos publicados por OMG al respecto son los relacionados con el conocido como *metamodelo del lenguaje de modelización UML* (UML Metamodel) (OMG, 2011b,c) y con un metametamodelo denominado *Meta Object Facility* (MOF) (OMG, 2011a). En ellos la definición de *class* es realmente confusa. Cabría preguntarse cuando se habla de *class* a qué *class* nos estamos refiriendo:

- **Especificación de UML 2.4 Infrastructure** (OMG, 2011b) incluye diferentes descripciones del término *class*:
  - *A class is a type that has objects as its instances*, en el *package* Core::Basic (apartado 10.2.1). Type es definido como *a named element that is used as the type for a typed element* (obsérvese que se trata de una definición vacía de contenido puesto que incluye dos veces el término definido).
  - *Class is a kind of classifier whose features are attributes and operations*, en el *package* Core:Constructs (apartado 11.3.2).
  - *Class has derived association that indicates how it may be extended through one or more stereotypes*, en el *package* Core:Profiles (apartado 12.1.1).
- Por otra parte la **Especificación de UML 2.4 Superstructure** (OMG, 2011c) incluye igualmente diferentes descripciones del término *class*:
  - *A class describes a set of objects that share the same specifications of features, constraints, and semantics*, en el *package* Kernel (apartado 7.3.7).
  - *A class extends the metaclass Class with the capability to have an internal structure and ports*, en los *packages* StructuredClasses e InternalStructures. (apartado 9.3.1).
  - *A class may be designated as active (i.e., each of its instances having its own thread of control) or passive i.e., each of its instances executing within the context of some other object)*, en el *package* Communications (apartado 13.3.8).

- *Class has derived association that indicates how it may be extended through one or more stereotypes* (apartado 18.3.1) (repetición del apartado 12.1.1 de (OMG, 2011b)).

- Finalmente la **Especificación de Meta Object Facility (MOF) 2.4** (OMG, 2011a) reutiliza la definición del término *class* en el *package* Kernel de UML.

En efecto, OMG parece indicar que el lector de estos documentos debe reconocer a qué clase realmente se refiere el término *class* profusamente utilizado en ellos con significados muy diferentes.[2] Esta ambigüedad en la utilización del término ha conducido en la práctica a encendidas discusiones sobre el verdadero significado de *class* en algunos casos (Henderson-Sellers and Gonzalez-Perez, 2006; Weisemöller and Schürr, 2008).

4.2.3. Sobreutilización del concepto de Instancia

El término *instance* utilizado igualmente en los mencionados documentos de OMG también ha sido sobreutilizado con diferentes significados. En versiones anteriores a UML 2.0 el término *instance* se empleó indistintamente para denominar tanto a las instancias de clases de usuario UML (el denominado nivel M0) como para referirse a la metaclase abstracta que las representa en el metamodelo de UML (nivel M2). Esta ambigüedad produjo mucha confusión y condujo igualmente a discusiones en la literatura sobre el significado real de *instance* (Atkinson and Kühne, 2001; Bézivin, 2001, 2005; Gonzalez-Perez and Henderson-Sellers, 2007; Seidewitz, 2003).

A partir de la versión 2 de UML se intenta resolver esta controversia introduciendo un nuevo concepto denominado *InstanceSpecification* definido como *a model element that represents an instance in a modeled system.* Esto ayudó a clarificar en parte el término *instance* pero no evitó su sobreuso tanto por OMG

[2]En la sección 18.1.1 de la versión 2.2 de UML (OMG, 2009b) los autores reconocen esa confusión afirmando:

> Thus, in this clause, when we mention *Class*, in most cases we are dealing with the meta-metaclass *Class*

Sin embargo, en la misma sección de la versión 2.4.1 de UML (OMG, 2011c) vigente en el momento de escritura, ese párrafo fue eliminado. Esa versión de UML fue el resultado del proceso de revisión del estándar al que hacíamos referencia en la introducción de este capítulo, y que fue publicada con posterioridad al envío a OMG de nuestro artículo (Rodriguez-Priego et al., 2010) en el que precisamente se mencionaban las dudas que planteaba dicho párrafo.

como por diversos autores (véase por ejemplo (Kühne, 2006a)). OMG introdujo una nueva clase *InstanceSpecification* con el fin de indicar sus relaciones con otras clases y puso énfasis en la importancia de no confundir dicha clase con las verdaderas instancias:

> It is important to keep in mind that InstanceSpecification is a model element and should not be confused with the dynamic element that it is modeling. Therefore, one should not expect the dynamic semantics of InstanceSpecification model elements in a model repository to conform to the semantics of the dynamic elements that they represent.

Sin embargo, UML y MOF no comparten las mismas *InstanceSpecification* lo que añade algo más de confusión a este problema. Estas *InstanceSpecification* pueden apreciarse en las figuras 4.14 y 4.15 (tomadas de (OMG, 2011a) y (OMG, 2011b) respectivamente), donde no pretendemos mostrar el detalle sino las apreciables diferencias que existen entre ellos.

Aunque la identificación de *entidad que se modeliza* con instancia de una clase se ha extendido en la literatura contagiada por la propuesta de OMG, no pocos autores (Bézivin, 2005; Gitzel and Hildenbrand, 2005; Gonzalez-Perez and Henderson-Sellers, 2007; Ober and Prinz, 2006) indican que este enfoque es demasiado restrictivo debido a las diferencias semánticas existentes entre los distintos tipos de instanciación: 1) metamodelo y modelo, 2) modelo y sistema que se modeliza 3) modelo y resultado de una fórmula que es generada por una gramática de cálculo de predicados (Ober and Prinz, 2006). Uno de los intentos más conocidos de diferenciar estos tipos es el de (Bézivin, 2005) quien advierte del peligro de usar la relación *instance-of* entre un modelo y su metamodelo y propone la denominación *conforms-to* o *c2* en su lugar, en un intento de diferenciar los diferentes significados de *instance* en distintos niveles de abstracción. De lo anterior se concluye que cuando se identifica un modelo con una clase y el sistema modelizado con una instancia se generan multitud de problemas conceptuales. Nótese que a la vista de esta situación y tal como hemos indicado se confirma que el problema de fondo es no reconocer que la orientación a objetos no es la base de la modelización sino una forma de implementarla. Un metamodelo podría ser igualmente implementado usando tablas relacionales y sus correspondientes modelos mediante tuplas de esas tablas (Aß mann et al., 2006; Thomas, 2004). En tal caso, si el enfoque de Entidad-Relación hubiese sido la base de la modelización la discusión, y la confusión, podría haber involucrado términos como *entity-type*,

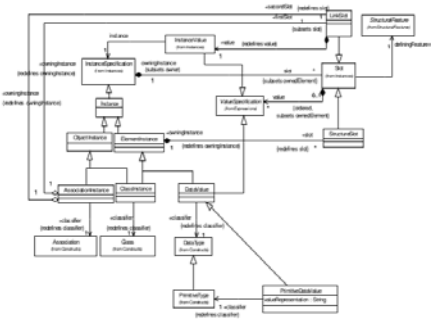


Figura 4.14: *InstanceSpecification* en MOF (tomada de (OMG, 2011a))

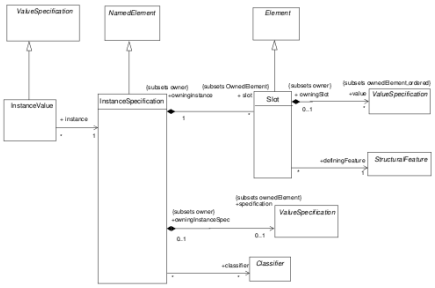


Figura 4.15: *InstanceSpecification* en UML (tomada de (OMG, 2011b))

entity, *association-type*, etc. Nuevamente, el homomorfismo entre esos conceptos y los correspondientes de la modelización hubiese sido el origen de la confusión. Para resolver esta situación, algunos autores (véase por ejemplo (Kurtev, 2007)), proponen que para comprender realmente los conceptos de modelo y metamodelo la discusión debería ser llevada al plano ontológico.

4.2.4. Carencia de diagramas orientados a objetos

Paradójicamente la orientación a objetos de UML no garantiza que sea el lenguaje más adecuado para modelizar sistemas orientados a objetos. El metamodelo de UML es efectivamente orientado a objetos porque está descrito mediante clases, pero no existen diagramas de modelización de UML que encapsulen la estructura y el comportamiento de un sistema tal como la propia teoría de la orientación a objetos establece como uno de sus principios básicos. Por el contrario, UML permite expresar mediante diagramas la estructura y el comportamiento por separado con lo que la visión del sistema completo no siempre es explícita (Merunka, 2003). El problema de fondo de esta limitación es que UML carece de una definición semántica precisa que permita delimitar qué elementos de un tipo de diagrama pueden ser combinados con elementos de otro tipo de diagrama. Una consecuencia de esta indefinición es que algunas herramientas permiten combinar elementos de diferentes tipos de diagramas y otras no. Esto ha llevado a algunos autores a estudiar la consistencia que tendría la combinación entre diagramas de estructura y comportamiento en UML (véase por ejemplo (Martínez and Álvarez, 2005)).

Kuske et al. (2009) van más allá y proponen un nuevo tipo de diagrama UML que denominan *Integrated Diagram* y que sería el resultado de la combinación de

diagramas de clase y de estado.

En la práctica sin embargo, tal como se indicó en la sección 2.2.2 los diagramas más utilizados en UML son los diagramas de clases y además en su versión más básica (sin apenas atributos u operaciones). De hecho algunos autores (véase por ejemplo (Gitzel et al., 2007)) concluyen que incluso cuando construyen modelos de alto nivel no necesitan incluir métodos en sus modelos. Quienes los incluyen se limitan en la mayoría de los casos a los métodos de asignación y consulta de atributos (*set* y *get*) más que a la representación del comportamiento de la clase. En definitiva, podríamos concluir que aunque UML sigue un enfoque orientado a objetos, los modelos creados con diagramas de clases UML son más una representación conceptual de atributos y relaciones que una representación de clases como encapsulación de estructura y comportamiento en el sentido de la orientación a objetos.

4.2.5. Complejidad de modelo frente a sistema modelizado

En la sección 4.1.1 indicamos que un modelo es una simplificación de la realidad, o dicho de otro modo, un modelo no debería ser más complejo que la entidad que modeliza. Esta característica junto a la de que un modelo puede reemplazar a su sistema modelizado para un propósito determinado implica que si el modelo es más complejo entonces sería más práctico utilizar directamente el sistema modelizado. Resulta evidente que las especificaciones orientadas a objetos propuestas por OMG que están relacionadas con la modelización, no cumplen la regla de la simplicidad. Así por ejemplo, las especificaciones del metamodelo de UML y el metametamodelo MOF no han tenido un volumen inferior a las mil páginas desde sus primeras versiones publicadas hace unos 10 años hasta las versiones actuales. De un análisis de estos documentos puede observarse que la complejidad es debida principalmente a las siguientes razones:

- Contienen la descripción de un número elevado de conceptos especificados en forma de clases.
- Hay redundancia entre los diferentes documentos, repitiéndose las mismas descripciones y duplicando conceptos con diferentes significados, tal como se ha indicado para el caso de clase e instancia.
- La jerarquía de herencia de clases es un auténtico laberinto en el que para

comprender las características de una clase hay que navegar por las interminables relaciones entre clases (France et al., 2006).

- La semántica asociada a cada una de las descripciones es expresada en lenguaje natural, a veces de forma muy extensa e incomprensible y a través de una serie de restricciones expresadas en el lenguaje OCL (OMG, 2012a).
- Las clases se agrupan en *packages* cuyas relaciones son en sí mismas complejas por la utilización de mecanismos de combinación (*package merge*) e importación de *packages* que son compartidos entre UML y MOF.

A mayor complejidad, mayor propensión a cometer errores. Algunos autores han analizado la consistencia de UML utilizando diversas técnicas formales y apoyándose en herramientas automatizadas para estudiar ciertos aspectos de UML. El resultado de estos estudios ha puesto de manifiesto errores e inconsistencias a lo largo de las diferentes versiones del metamodelo de UML, véase por ejemplo (Dingel et al., 2008) sobre combinación de paquetes (*package merge*), (Milicev, 2007) sobre asociaciones, (Akehurst et al., 2008; Shan and Zhu, 2008) sobre la semántica de UML en general, (Buttner and Gogolla, 2004) sobre generalización y sobrecarga, y (Fuentes et al., 2003) sobre UML 1.x.

Mención especial requiere el concepto de combinación de paquetes *package merge* que es uno de los mecanismos en los que OMG basa la mayor parte de la organización de las clases en los metamodelos de UML y MOF. El mecanismo *package merge*, es definido como *a directed relationship between two packages, that indicates that the contents of the two packages are to be combined* (OMG, 2011b). Tras esta aparentemente simple definición se esconde un mecanismo realmente complejo que requiere una extensiva explicación de 9 páginas (OMG, 2011b, pp. 163-171). La semántica de *package merge* podría considerarse que es similar a la herencia de clases y su aplicación no está exenta de problemas, tal como han puesto de manifiesto algunos autores (Dingel et al., 2008; Pottinger and Bernstein, 2003). Su principal problema es que produce algunas inconsistencias e indeterminaciones en su aplicación. Algunos autores han analizado su definición formal a nivel de modelo (Bottoni et al., 2006) mientras que otros autores (Barbero et al., 2007) proponen alternativas de menor complejidad. En nuestra opinión, *package merge* añade aún más complejidad a la ya de por sí compleja estructura de clases, puesto que no se hace explícito el resultado de la combinación de paquetes. Por ejemplo, si existe una clase del mismo nombre en dos paquetes que se combinan el resultado es una nueva clase (con el mismo nombre) que *hereda* los atributos y

métodos de ambas. Piénsese por tanto que habría que añadir aún más definiciones de clase, que surgen del resultado de diferentes combinaciones de paquetes (véase por ejemplo la figura 4.17).

Todos estos problemas son una confirmación de la opinión expresada por France et al. (2006) sobre la comprensibilidad de la semántica de UML, que se refiere al UML Metamodel como UML *Metamuddle*, afirmando que debido a la complejidad del metamodelo de UML sólo mediante herramientas de consulta y extracción pueden obtenerse vistas simplificadas del mismo.

4.3. Dificultades sobre los niveles de la modelización

Hemos indicado que según la propuesta de OMG todo está definido mediante clase e instancia. Cuando las instancias de una clase son a la vez clases se habla de metaclases y si éstas son instancias de otra clase se la llama metametaclase y así sucesivamente. OMG entiende que cuando se instancian clases en clases éstas están en un nivel de abstracción inferior y propone lo que denomina la *jerarquía de metamodelo de cuatro niveles* como referencia para comprender estos saltos en el nivel de abstracción. El nivel más alto de la jerarquía, también denominado M3 es un metametamodelo y según (OMG, 2011b) su principal cometido es definir el lenguaje que permita especificar un metamodelo. Este nivel, además se define a sí mismo, evitando una interminable jerarquía de niveles superiores. El siguiente nivel M2, está formado por metamodelos y de manera similar su función, según OMG, es definir lenguajes para especificar modelos. El siguiente es M1, y define modelos que permiten a los usuarios modelizar en diferentes dominios. Finalmente el nivel M0 contiene las instancias de los modelos definidos en M1. La figura 4.16, ilustra esta jerarquía donde puede verse que OMG sitúa MOF en el nivel M3 y UML en el nivel M2. Esta estructura está inspirada en otras propuestas como ANSI IRDS y EIA/CDIF (Flatscher, 2002).

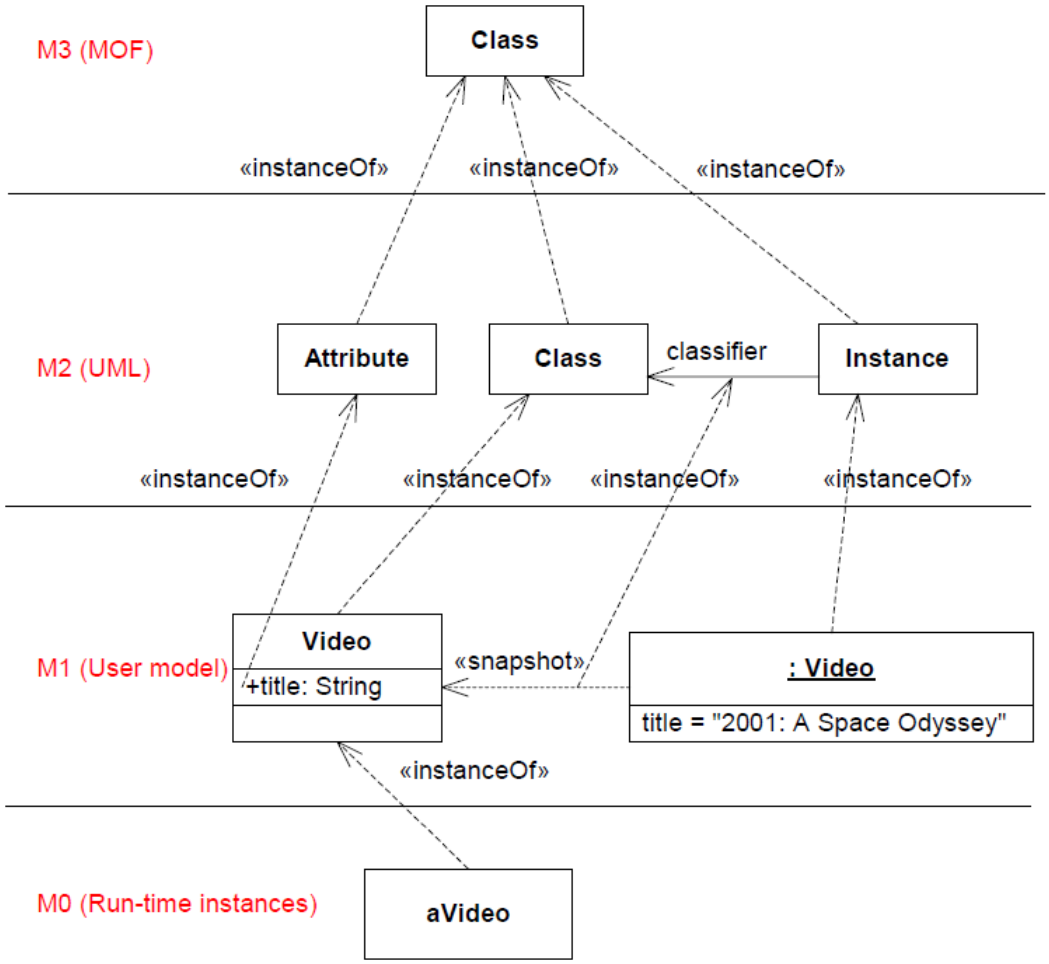


Figura 4.16: Ejemplo de jerarquía de 4 niveles de OMG (tomada de (OMG, 2011b))

OMG agrupa las clases en *packages* y éstos en lo que denomina *libraries*. El lector de las especificaciones OMG esperaría que todas las clases de cada nivel de abstracción estuviesen agrupadas en *libraries*, es decir, *libraries* de M3, M2, etc. con el fin de distinguir fácilmente si se trata de instancias, clases, metaclases y metametaclases. Sin embargo, la estructura de *libraries* de UML y MOF está constituida por un conjunto que forma a su vez una jerarquía paralela que nada tiene que ver con la jerarquía de cuatro niveles. La figura 4.17, muestra como clases del nivel M3 (MOF) utilizan clases del nivel M2 (UML) mediante los dos tipos de mecanismos de relaciones entre *packages* ya mencionados: *import* y *merge*.

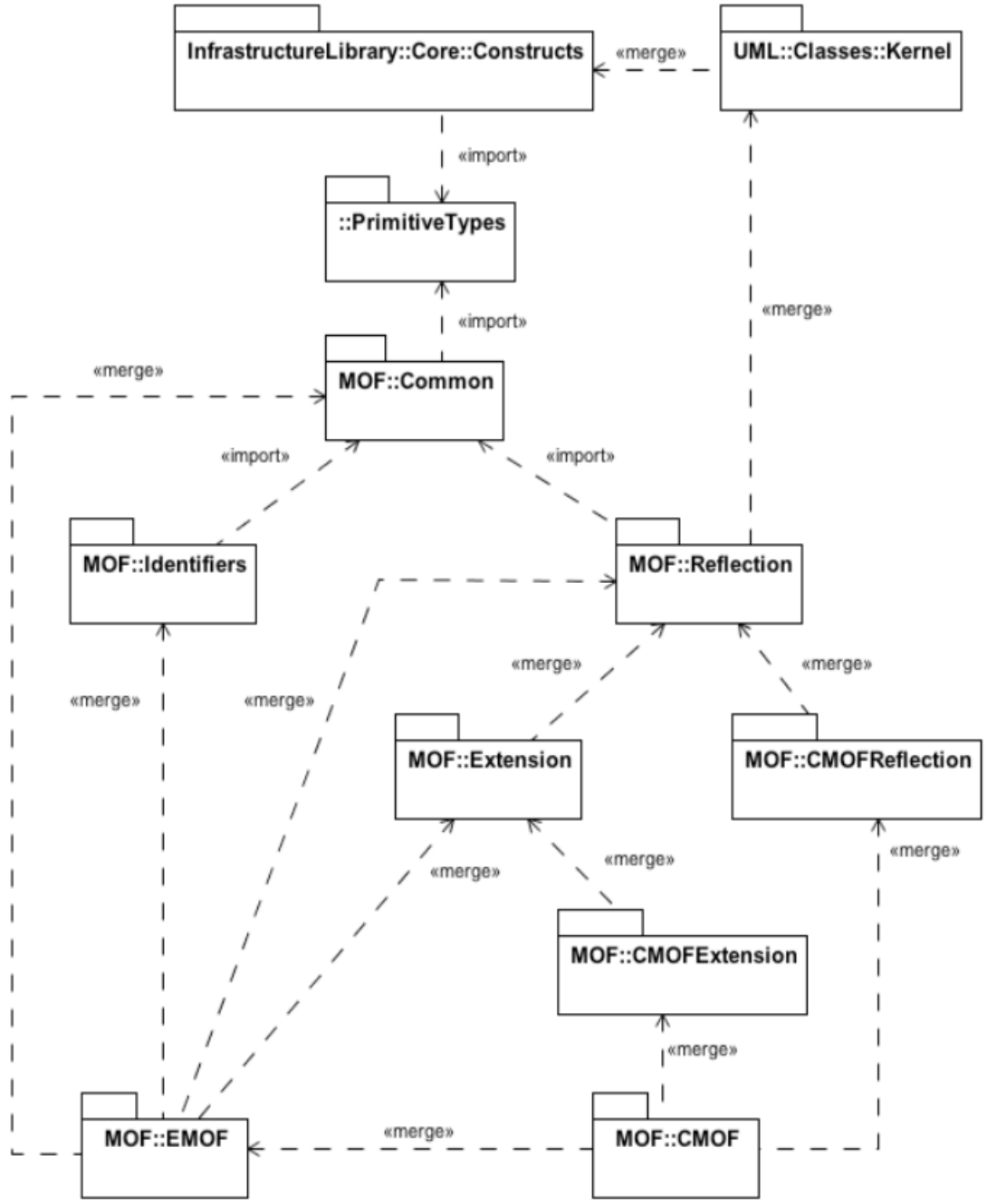


Figura 4.17: Reutilización de *libraries* de UML en MOF (tomada de (OMG, 2011a))

El resultado es, como mínimo, bastante confuso. Al compartir UML y MOF gran parte de las clases de una misma *library* (*InfrastructureLibrary* en la figura 4.17) las instancias de las clases de ésta dan lugar a clases de los niveles M3 y M2. La pregunta obvia es ¿no deberían ser las clases de esa *library* las que conformen

el nivel M3, cuya característica principal es precisamente la de dar lugar a M2 y M3? (recuérdese que M3 es reflexivo). Los autores de las especificaciones resuelven este problema indicando que hay efectivamente otra jerarquía formada por la estructura de los *packages* (OMG, 2011b):

> (...) the package structure is layered to separate the *metalanguage* core constructs from the higher-level constructs that use them

Se introduce por tanto otro concepto, el *metalenguaje*, con la intención de diferenciarlo de *metamodelo*. Sin embargo, es difícil no confundirse puesto que metamodelos y metalenguajes comparten los mismos conceptos (*package*, *class*,.. etc.). Nótese que tal como indicábamos en la sección 4.2 la terminología utilizada (*class*, *package*, *library*, *import*, etc.) está muy alejada de los términos utilizados en la modelización y recuerda a la utilizada en los lenguajes de programación como Java o Ada, reafirmándose la percepción de que se trata más de la descripción de una implementación cercana a los lenguajes de programación que una especificación de modelos o lenguajes.

El resultado de este confuso enfoque es una multitud de discusiones en la literatura en relación con los niveles de abstracción, la metamodelización y su relación con los (meta)lenguajes, cuyo origen es nuevamente consecuencia de la sobreutilización del concepto de instancia y clase. Así algunos autores (Atkinson and Kühne, 2003; Gašević et al., 2007) proponen distinguir entre dos tipos de instanciación: uno denominado ontológico y otro denominado lingüístico ahondando en la idea de OMG de una doble jerarquía. Otros autores proponen distinguir entre instanciación estricta (cada nivel sólo puede instanciar al nivel inmediatamente inferior) o instanciación profunda (un nivel puede instanciar también a niveles inferiores) (Atkinson and Kühne, 2001; Gitzel et al., 2007; Varró and Pataricza, 2001) dando lugar a un enfoque muy diferente denominado metamodelización multinivel, aunque a costa de una mayor complejidad. (Gitzel and Hildenbrand, 2005) explora un camino en cierto modo opuesto diferenciando entre jerarquías lineales y no lineales (aquellas en las que la instanciación también puede producirse en el mismo nivel). Otra propuesta, ya mencionada, es definir relaciones diferentes a la de instanciación, por ejemplo conforms-to, represented-by, etc. (Bézivin, 2005; Gonzalez-Perez and Henderson-Sellers, 2007).

4.3.1. Necesidad de un modelo raíz mínimo

El carácter reflexivo del nivel superior (M3) de la jerarquía es también objeto de discusión. Con la idea de que un metametamodelo debe ser el menos complejo de la jerarquía y sin abandonar la propuesta de 4 niveles, (Jouault and Bézivin, 2006) propone una alternativa a MOF algo más simple: 14 clases en lugar de las 24 clases de MOF. (Seidewitz, 2003) analiza los problemas derivados de la aplicación de un modelo en el nivel M3, definiendo el concepto de *metamodelo reflexivo mínimo* como aquél en el que cualquier sentencia de dicho modelo puede ser representada en términos de elementos de sí mismo. Igualmente (Gitzel and Hildenbrand, 2005) discute las ventajas e inconvenientes entre dos posibles tipos de metametamodelo: el recursivo y el axiomático. (Gonzalez-Perez and Henderson-Sellers, 2007) analiza los problemas derivados de que MOF no sea realmente un modelo mínimo y afirmando que un verdadero modelo recursivo debería ser mínimo.

A pesar de que éstos y otros autores han expresado la necesidad de que el modelo raíz (situado en la parte superior de la jerarquía) debería ser mínimo, es decir lo más simple posible, no conocemos ninguna propuesta de modelo raíz mínimo (recursivo o axiomático) que ayude a evitar estos problemas, clarifique la teoría de la modelización y constituya el verdadero núcleo de la modelización más allá de la confusa propuesta de OMG. Se trata, sin lugar a dudas de un área de investigación abierta no exenta de dificultades. (Gonzalez-Perez and Henderson-Sellers, 2007) sin llegar a formalizar una propuesta, sugieren el uso de conceptos universales:

> (...) any SUS can be represented as a collection of *things* and *connections between things*. In other words, as a graph of nodes and arcs.

Sin embargo, concluyen que incluso con este enfoque tan simple, existen problemas y muestran un ejemplo en el cual no queda claro si un determinado concepto puede ser considerado como nodo o enlace: en un sistema bibliográfico podrían distinguirse como *nodes* Libros, Lectores y Préstamos pero también podría interpretarse que Préstamo no es un *node* sino un *arc* entre Libro y Lector.

4.3.2. Bidireccionalidad entre modelo y entidad modelizada

Además de los problemas inter/intra/multinivel que plantea la propuesta de OMG al usar la instanciación como medio para bajar en el nivel de abstracción, existe

otro problema que afecta a una de las características esenciales de la modelización: el carácter descriptivo o prescriptivo de los modelos. En la sección 4.1.1 hemos indicado que diversos autores definen modelo como una descripción o como una especificación de un sistema/realidad/original. Básicamente en su modalidad descriptiva, un modelo permite representar a una entidad *existente* a la que modeliza para poder alcanzar un propósito de una manera más simple, mientras que en su modalidad prescriptiva un modelo *existente* permite obtener una entidad nueva que es igualmente modelizada por éste. Esta doble relación de los modelos con las entidades que modeliza ha sido tratada por la literatura con diferentes nombres. Así la relación de descripción, es llamada *represented-by* (Bézivin, 2005), *representation-of* (Favre and Nguyen, 2004; Jouault et al., 2009) y los correspondientes modelos *descriptive models* (Bézivin, 2005; Favre, 2004a; Seidewitz, 2003), *backward-looking models* (Gonzalez-Perez and Henderson-Sellers, 2007), *model as copy* (Sánchez et al., 2009) o *reverse engineering* (Jouault et al., 2009).

Del mismo modo para la especificación se utilizan los nombres de *prescriptive models* (Bézivin, 2005), *specification models* (Seidewitz, 2003), *forward-looking models* (Gonzalez-Perez and Henderson-Sellers, 2007), *model as original* (Sánchez et al., 2009) o *forward engineering* (Jouault et al., 2009).

La mayoría de los autores resaltan dos aspectos de los conceptos de descripción y especificación que son relevantes para entender el problema que se plantea con el enfoque OMG:

- El aspecto descriptivo o prescriptivo de un modelo indica cómo es usado el modelo, no para etiquetar el modelo en sí mismo. Dicho de otro modo, según la literatura, un mismo modelo puede verse desde su óptica descriptiva o prescriptiva.
- El punto clave que diferencia la prescripción de la descripción es quién existe primero. En la descripción, la entidad modelizada existe previamente al modelo mientras que en la especificación ocurre justo al contrario.

Teniendo en cuenta lo anterior analicemos cómo aborda esta cuestión la propuesta de OMG. La respuesta la encontramos rápidamente al observar la figura 4.16: el sentido de las flechas entre un modelo y la entidad que modeliza siempre es de la entidad modelizada al modelo, o según OMG, de la instancia a la clase. El significado de esa representación como una línea terminada en flecha es explicada en (OMG, 2011b):

> The following conventions are adopted for all metamodel diagrams throug-

> hout this specification: An association with one end marked by a navigability arrow means that the association is navigable in the direction of that end. (...)
>
> Navigability means instances participating in links at runtime (instances of an association) can be accessed efficiently from instances participating in links at the other ends of the association. The precise mechanism by which such access is achieved is implementation specific. If an end is not navigable, access from the other ends may or may not be possible, and if it is, it might not be efficient.

Por tanto, dada una instancia (que siempre es generada a partir de una clase mediante instanciación) podemos *conocer* su clase pero no a la inversa. Esto es debido a que no existe, desde el punto de vista de OMG, la operación inversa, es decir, que primero exista la instancia y a partir de ella sea obtenida la clase mediante una operación inversa a *instantiation*, que podría llamarse, en coherencia con la nomenclatura de OMG, *class*ification. Esta operación se representaría de manera similar mediante una flecha de clase a instancia para indicar que ese tipo de clase (descriptiva) *conoce* a la instancia que está describiendo. En la práctica, la implementación de modelos UML mediante lenguajes orientados a objetos confirma esta característica de los modelos prescriptivos, puesto que mientras que sí suele existir una función del lenguaje para obtener la clase de un objeto no existe una función que devuelva todas las instancias que ha generado una clase.

En definitiva, la bidireccionalidad de la relación entre modelo y entidad modelizada, a pesar de ser una de las características más importantes de la modelización, no es contemplada en la propuesta de OMG que sólo se centra en los modelos prescriptivos. Nótese que una revisión detallada de MOF puede hacernos pensar que esa bidireccionalidad ha sido contemplada en algunas situaciones (una clase factoría que conoce a sus instancias). Sin embargo, realmente no se da esa situación puesto que incluso en ese caso la instancia fue creada a partir de dicha clase y no al contrario.

Esta carencia del enfoque OMG deja fuera de su alcance un tipo más de modelos que son a la vez descriptivos y prescriptivos. Según (Ludewig, 2003) en este tipo de modelos las modificaciones son aplicadas al modelo más que al sistema modelizado y cuando esas modificaciones parecen dar el resultado deseado entonces son aplicadas al original. Una propuesta similar, denominada *models at runtime* es presentada por (Jouault et al., 2009). La complejidad de esta solución radica principalmente en dos aspectos: 1) el modelo debe mantener los cambios

en el sistema modelizado y viceversa y 2) el sistema modelizado necesita tener acceso al modelo. No es sorprendente que en este escenario los modelos no son necesariamente representados por un conjunto de objetos al estilo OMG, sino que en su lugar los autores conforman un repositorio de elementos de modelización que constituyen en sí mismos un sistema que interactúa con los sistemas a los que especifican o describen.

4.3.3. Cardinalidad entre modelo y entidad modelizada

Existe un aspecto más en la relación entre un modelo y la entidad modelizada que ha merecido la atención de la literatura. Se trata de la cardinalidad de dicha relación. Kühne (2006a) distingue dos tipos de modelos en relación con dicha cardinalidad: *token models* y *type models*. En los token models la cardinalidad es de 1 a 1, es decir, un token model captura aspectos singulares (como opuestos a universales) de la entidad modelizada. Los type models por el contrario tienen una cardinalidad de 1 a n, es decir los modelos capturan propiedades universales (Kühne, 2006a). Kühne (2006a) indica que los token models pueden ser creados en UML mediante diagramas de objeto, en los que la correspondencia de las propiedades de la entidad modelizada no son con los atributos de una clase sino con valores concretos de esos atributos, pero también con diagramas de clase en algunos casos. En la misma línea, Gonzalez-Perez and Henderson-Sellers (2007) distingue entre 1) isotypical mappings como aquellos que mapean un modelo con un sistema bajo estudio, 2) prototypical mappings que mapean un modelo con un conjunto de sistemas dados como ejemplo y 3) metatypical mappings que es similar a protypical mappings pero caracterizados declarativamente (Gonzalez-Perez and Henderson-Sellers, 2007).

Como veremos más adelante, la cardinalidad es un aspecto de la modelización cuya comprensión permite entender mejor los conceptos de modelo y original. Aunque como indicaba Kühne (2006a) puede realizarse una interpretación de la cardinalidad en UML, no es un aspecto que sea abordado directamente por el metamodelo de UML y esa interpretación no está exenta de problemas. Efectivamente si ya plantea bastantes inconvenientes identificar modelo con clase, aún más confusión añadiría el hecho de que hubiera un tipo de modelos que se correspondiese con clases y otro con objetos, especialmente porque si en un diagrama de objetos se están representando muestras de instancias de una clase no se entendería bien que siendo instancias se equiparasen con modelos en lugar de con

entidades modelizadas.

4.4. Dificultades sobre Metamodelización

4.4.1. Estereotipos UML

Una de las pretensiones de UML era la de servir como lenguaje universal de modelización unificando las técnicas existentes hasta antes de su creación. El resultado ha sido un lenguaje formado por una lista de sublenguajes aplicables en diferentes dominios. Sin embargo, su complejidad, a la que ya hemos hecho referencia, su redundancia y la existencia de funcionalidades de poco uso hacen que en la práctica muchos usuarios u organizaciones sólo usen una parte básica (más simple de UML) y busquen un complemento en lenguajes específicos realizados a medida (DSL). OMG consciente de este problema, decidió buscar una solución para crear DSLs a partir de UML y MOF.

Siguiendo el planteamiento de OMG, un DSL al ser un *lenguaje de modelización* tendría su metamodelo en el nivel M2 que es el nivel donde está el metamodelo de UML. El metamodelo de UML no podría servir para definir un DSL puesto que está restringido a los modelos estándar UML. Por tanto deberíamos ascender al nivel M3 para poder definir el metamodelo de un DSL, lo cual en la arquitectura de 4 niveles de OMG nos llevaría a MOF. Llegados a este punto, OMG se encontró con un complicado problema: mientras existen multitud de herramientas que permiten crear modelos UML de una manera gráfica e interactiva, no existen herramientas similares para crear metamodelos a partir de MOF. La idea que se le ocurrió para resolver este problema fue crear un nuevo tipo de diagrama al que denominó *Profiles Diagram* en el que en lugar de crearse modelos se creasen metamodelos. A los elementos principales de esos diagramas les denominó *stereotypes* y, como no podía ser de otra forma, definió un nuevo tipo de clase para dar cabida a este extraño elemento que debería ser de nivel M3 pero que aparece en un modelo de nivel M2.

Esto da lugar a nuevos problemas. Efectivamente *stereotype* es definido como una especialización de otro elemento que, para más confusión, también lo denominan *class* y que abarca todos los elementos de UML (por ejemplo, *Actor*, *Activity*, etc.) del nivel M2 y por lo tanto, debería estar definido en el nivel M3, es decir debería ser una clase de MOF. Como esto no es posible, recurren

al *truco* de usar la implementación de *MOF::Class* definida en la librería *UML Infrastructure Library* estableciendo que *Profiles::Class* es una subclase de *InfrastructureLibrary::Core::Constructs::Class*.

Además de una nueva clase, se define una nueva relación de asociación denominada *extension association* que enlaza *stereotype* (que sería de nivel M3) con una *Profile::Class* (también de nivel M3). Como es de esperar, y de manera similar a lo realizado con las clases, *extension* es una especialización de *InfrastructureLibrary::Core::Constructs::Association*. Sin embargo, a pesar que *stereotype* es una especialización de *Profile::Class*, no es posible extender un *stereotype* con otro *stereotype*, aunque sí se permite, con algunas restricciones, que *stereotype* forme parte de asociaciones (OMG, 2011b, Sec. 12.1.7). Nótese que en virtud de lo anterior, se permiten algunas relaciones sin sentido aparente como la definición de un *stereotype* que por ejemplo extienda *Actor* y *State* y que además tenga una relación de asociación con otro *stereotype* que extienda por ejemplo *Activity*. Los autores de la especificación dejan en manos del usuario de estas definiciones que la semántica de los metamodelos obtenidos de este modo sea consistente (OMG, 2011b, Sec.12):

> A profile must provide mechanisms for specializing a reference metamodel (such as a set of UML packages) in such a way that the specialized semantics do not contradict the semantics of the reference metamodel.

Aparentemente no hay nada que impida hacer ese uso de clases y relaciones de M3 en M2 (de hecho, tal como hemos indicado OMG usa esa práctica reiteradamente haciendo que UML y MOF compartan paquetes en una jerarquía paralela). Sin embargo, como es de esperar, surgen problemas: puesto que se ha forzado la definición de conceptos M3 en M2 no es posible realizar determinadas operaciones que sólo tienen sentido para M3. Sirva como ejemplo el de la serialización: OMG define XMI (MOF, 2011) como la forma estándar de exportar a un documento XML un modelo realizado mediante MOF. Puesto que los *UML Profiles* no están realizados mediante MOF no era posible exportarlos a XMI para que puedan ser intercambiados entre diferentes herramientas que soporten XMI. La solución es nuevamente lo que se podría nombrar como un apaño: los autores de UML definen una correspondencia entre *stereotypes* y elementos de MOF en (OMG, 2011b, sec. 12.1.7). Este uso no trivial de un concepto M3 en M2 ha dado lugar a no pocas discusiones y críticas sobre la semántica de los *UML Profiles* en la literatura (Henderson-Sellers and Gonzalez-Perez, 2006; Weisemöller and Schürr,

2008).

Desgraciadamente, a pesar de este esfuerzo para favorecer la interoperabilidad de *UML Profiles* mediante XMI, dicha operabilidad aún es difícil de alcanzar por diferentes motivos:

- Existen distintas versiones de XMI y de UML incompatibles entre sí.
- El estándar XMI no considera los datos relacionados con la representación de un diagrama, por lo que esta funcionalidad sólo está cubierta por extensiones propietarias incompatibles entre sí.
- Las herramientas no suelen implementar el estándar en su totalidad debido a la complejidad de las especificaciones.
- OMG publicó en 2006 el estándar UMLDI (UML Diagram Interchange) con el propósito de (OMG, 2006b):

  > storage and exchange of information pertaining to the layout of UML Models

  Sin embargo, en la práctica, tal como es admitido por la propia OMG (OMG, 2007, p. 21):

  > the DI standard itself is not precise enough to enable consistent exchange of diagrams between different tools

  En el momento de escritura, este estándar está sujeto a un proceso de revisión que permita su adopción por los fabricantes de herramientas UML.

En definitiva, aunque el objetivo inicial es loable, el resultado final es que con los *UML Profiles* OMG opta por una solución que añade aún más complejidad al complejo metamodelo de UML y contribuye aún más a la confusión entre el metamodelo de UML y MOF provocado por la compartición de *libraries* en las especificaciones de estos estándares.

Además de esto, al ofrecerse los *UML Profiles* como una alternativa a MOF para crear metamodelos cabría preguntarse cuál de las dos opciones sería más conveniente elegir en un caso dado. OMG responde a esa cuestión indicando que:

> There is no simple answer for when you should create a new metamodel and when you instead should create a new profile. (OMG, 2011b, Sec.12)

es decir, existe una respuesta a esa pregunta pero es complicada. Sin embargo, los diseñadores aún en el momento de escritura, se plantean esa cuestión puesto que los enfoques de MOF y *UML Profile* son tan diferentes que una elección equivocada podría comprometer considerablemente el resultado final.

4.4.2. Herencia y metamodelización

Desde el punto de vista orientado a objetos de OMG hay dos conceptos que aparecen de manera sistemática en el contexto de la modelización y que en muchas ocasiones dan lugar a confusiones. Se trata de la herencia y la metamodelización entendida por algunos autores como un resultado de la instanciación de modelos en otros modelos.

Herencia: la herencia es un concepto genuinamente orientado a objetos y trasladado al contexto de la modelización implica que, al igual que una clase (denominada subclase) puede heredar de otra (superclase), un submodelo puede heredar de un supermodelo. En lenguaje natural esta derivación se expresa mediante el término **es un** (*is a*).

Por ejemplo, asumiendo que existe un modelo de *gato* y un modelo de *animal* podríamos decir que *gato* ***es un*** *animal*. Por otro lado, supongamos que existe un gato que llamaremos *Mini*. En lenguaje natural también se dice que *Mini* ***es un*** *gato*. Como puede observarse la expresión ***es un*** en *gato* ***es un*** *animal* y en *Mini* ***es un*** *gato* tienen significados muy diferentes: en el primero se trata de herencia y en el segundo de instanciación. Aunque un creador de modelos podría confundirse inicialmente pensando que la relación entre *Mini* y *gato* es de herencia, en este ejemplo saldría fácilmente de su error al observar que *Mini* no es un modelo sino una instancia y la relación de herencia, según la teoría de la orientación a objetos, siempre es entre clases. Sin embargo, que la herencia sólo sea entre clases no es suficiente para diferenciarla de la instanciación puesto que ésta también puede existir entre clases cuando una clase (calificada como metaclase) se instancia en otra clase.

Afortunadamente podemos ayudarnos de dos reglas bien conocidas que siempre han de cumplirse para que una relación sea calificada de herencia y que pueden enunciarse de la siguiente forma:

1. Las características finales de la subclase siempre son el resultado de la unión entre el conjunto de características inicial de la subclase y el conjunto de características de la superclase. En el ejemplo, las características de un gato son la unión de las características de un animal y de las características específicas de los gatos.
2. El conjunto de instancias de la subclase siempre es un subconjunto del conjunto de instancias de la superclase. Esto en la práctica quiere decir que una instancia de la subclase es también una instancia de la superclase en todos

> los sentidos. O dicho de otra forma: todas las afirmaciones que podamos hacer sobre la superclase también podremos hacerlas sobre la subclase.

Una importante consecuencia de ambas reglas es que la herencia siempre es transitiva como lo es la inclusión de conjuntos (tanto de instancias como de características).

La segunda regla es importante porque generalmente es la principal *prueba de fuego* para saber si estamos ante un caso de herencia. Su no aplicación ha sido la causa de no pocas confusiones sobre la herencia en los primeros años de la orientación a objetos. Así por ejemplo, en (Atkinson et al., 1990) se afirma erróneamente que la clase Cubo es una subclase de Rectángulo. Ciertamente se cumple la primera regla puesto que las características de Cubo (altura, ancho y largo) pueden ser el resultado de la unión de las características de Rectángulo (Ancho y Largo) y la característica específica de Cubo (Alto). Sin embargo, no se cumple la regla 2 puesto que una instancia de Cubo no puede considerarse una instancia de Rectángulo. Así en el primer ejemplo, la afirmación *todos los animales nacen y mueren* también puede decirse de los gatos por ser animales. Sin embargo, en este ejemplo la sentencia *todos los rectángulos son planos* no puede decirse sobre los cubos. (Duntemann and Marinacci, 1990) comete un error similar indicando que la clase círculo es una subclase de la clase punto al afirmar erróneamente que:

> A circle differs from a point only in that a circle has a radius.

El origen de todas estas antiguas confusiones con la herencia procede del no cumplimiento de la segunda regla y de que en realidad la relación existente entre las supuestas sub y super clases no es de herencia sino de composición. En efecto, un círculo *está compuesto* por un punto y un radio y un cubo está compuesto de un rectángulo y una altura. Como apuntaba (Thomas, 1989) era necesario un mecanismo de *part-of* en la orientación a objetos para clarificar que se trataba de otra relación y evitar las confusiones producidas porque *part-of* cumple la regla 1 pero no la regla 2 (cuya comprobación siempre es olvidada).

UML incluyó dos tipos de asociaciones que dan respuesta a la necesidad de representar relaciones *part-of*: la *agregación* (donde la existencia de las *partes* es independiente de la del *todo*) y *composición* (donde las *partes* no pueden existir independientemente del *todo*). Sin embargo, el metamodelo de la orientación a objetos sigue sin contemplar este tipo de relación y en la práctica, la mayoría de los lenguajes de programación orientados a objetos actuales incluyen una palabra

reservada específica para herencia pero no para *part-of*.

Metamodelización: básicamente el proceso de *instanciación* consiste en la generación de un objeto a partir de una plantilla de características (atributos y operaciones) definidos por una clase. Instanciar por tanto, es obtener una entidad concreta (la instancia) a partir de una más general (la clase) asignando valores concretos a cada una de las características especificadas por la clase. Cuando la instanciación de una clase tiene como resultado otra clase se dice que la primera es una metaclase. Si a su vez la metaclase fuese una instancia de otra clase, ésta se denominaría metametaclase y así sucesivamente.

Cabe preguntarse si las reglas enunciadas sobre la herencia se cumplen también en la instanciación de clases y la respuesta es que:

1. No podemos afirmar que las características de una clase siempre sean el resultado de la unión de las características de su metaclase y sus características específicas.
2. No podemos afirmar que el conjunto de instancias de una clase sea un subconjunto de las instancias de la metaclase. Algunos autores (Atkinson and Kühne, 2002b; Atkinson et al., 2000; Kühne, 2006a) han estudiado este problema concluyendo que la relación *instance-of* no es transitiva en contraposición a la relación *is-a* que como hemos visto sí lo es.

Dicho esto parece claro que metamodelización, como un caso particular de instanciación, y herencia son mecanismos de naturaleza muy diferente y fácilmente diferenciables. Sin embargo, la literatura ha tratado extensamente algunos problemas relativos a estos dos conceptos demostrándose que en contra de lo que parece inicialmente sí existe una enorme confusión sobre ellos.

En primer lugar, como ya se ha apuntado (Bézivin, 2005) defiende que la relación entre un modelo y su metamodelo no es la de instanciación sino que se trata de un tipo de relación muy diferente que denomina *conforms-to*, es decir un modelo no es una instancia de su metamodelo sino que es conforme a él. Ya indicamos que esta distinción ha sido adoptada por un numeroso grupo de autores (Favre, 2006; Gargantini et al., 2009; Gašević et al., 2007; Karlsch, 2007; Moreno et al., 2008; Mu et al., 2010; Sánchez et al., 2009) quienes también consideran que instanciación y conformidad son conceptos muy diferentes. Sin embargo, otros autores y las propias especificaciones de OMG sobre UML y MOF (Aß mann et al., 2006; Kühne, 2006a; OMG, 2011b) mantienen que la obtención de una instancia a partir de un modelo sigue el mismo proceso que la obtención de un modelo a

partir de su metamodelo.

(Bézivin, 2005) va más allá, afirmando que la relación entre una instancia y su modelo no es *instance-of* como en orientación a objetos, sino que habría que denominarla *represented-by*, puesto que un sistema es representado por uno o varios modelos, haciendo hincapié por tanto en el aspecto descriptivo de los modelos. El mismo autor cambia posteriormente el sentido de la relación renombrándola *representation-of* (Jouault et al., 2009). Este cambio puede interpretarse como una evolución a un enfoque prescriptivo de los modelos.

(Aß mann et al., 2006) introduce más relaciones en un intento de realizar una clasificación que clarifique las similitudes y diferencias entre ellas, definiendo para ello una relación genérica denominada *similarity* que se subdivide en dos tipos de relaciones: *subset-of* e *is-represented-by* que a su vez se subdivide en *is-described-by* e *instance-of*. La relación *is-a* es un tipo de *subset-of* e *is-described-by*. Sin embargo, esta clasificación no incluye a la relación *conforms-to* y por tanto no ayuda a clarificar la confusión existente entre todas las relaciones.

Más recientemente Kühne (2009) aborda de nuevo el problema desde un punto de vista genérico comparando clasificación (que es un concepto cercano a instanciación) y generalización (herencia) poniendo de manifiesto que todavía existe un problema conceptual entre clasificación y generalización puesto que ambos conceptos siguen siendo aplicados incorrectamente. El análisis expuesto en Kühne (2009) pone de manifiesto que la confusión está relacionada con el lenguaje. Por ejemplo, se menciona el siguiente razonamiento para mostrar que la clasificación no es transitiva:

> Man is a species
> Socrates is a man
> Socrates is a species

El autor ilustra con este ejemplo que la instanciación no es transitiva. Sin embargo, se trata de una confusión lingüística puesto que la palabra *man* en la primera frase en realidad significa *Especie humana* (mankind) mientras que en la segunda significa *humano* (man) y utilizando esos términos correctos no podríamos construir un silogismo correcto. Efectivamente si, a la inversa intentáramos utilizar el término correcto (mankind) en la primera frase tendríamos que utilizarlo igualmente en la segunda frase para construir el silogismo pero veríamos que sería igualmente falso:

> Mankind is a species

> Socrates is a mankind
> Socrates is a species

A continuación (Kühne, 2009) se centra en las contradicciones que se observan en las especificaciones UML y menciona un problema que ya hemos indicado en la sección 4.3: la existencia de un paquete que parece actuar como nivel M3 pero que está por encima de MOF. En efecto, pone de manifiesto que UML es a la vez una instancia y una subclase del paquete Core (OMG, 2011b). Tras una serie de consideraciones indica que si fuese así constituiría una contradicción según el razonamiento que expone en el artículo. Sin embargo, finalmente concluye que no existe ningún problema puesto que:

> As a (surprising!) result, it is indeed possible to increase the utility of Core/MOF by using it both as a repository format (linguistic type) and a language definition supermodel (language definition supertype) while maintaining a clean structure free of inconsistencies.

y esto es posible debido a que según (Atkinson and Kühne, 2005):

> the OMG uses MOF and hence by implication the Core, both as a ontological type model and a linguistic type model.

Cabe preguntarse si ese doble carácter ontológico o lingüístico de UML/MOF/-Core no es tal y nuevamente se esté empleando una misma palabra *UML* para referirse a conceptos diferentes como son *UML Metamodel* y *UML Metalanguage*, tal como ocurría en el ejemplo anterior con la palabra *man*, y por tanto el principal origen de la confusión entre instanciación y herencia sea debido a las ambigüedades del lenguaje empleado.

No es casual que la mayor parte de las discusiones relacionadas con herencia e instanciación estén relacionadas con UML, puesto que en primer lugar, tal como señala (Buttner and Gogolla, 2004):

> the way the new semantics for subclassing and overriding is defined in the UML 2.0 specification is highly complex, scattered over many diagrams, constraints and additional operations sections and is very difficult to understand.

y en segundo lugar, la confusa definición de *stereotype* entre los niveles M2 y M3, ya mencionada en la sección 4.4.1, ha dado lugar a usos en M1 en los que el mecanismo correcto que ha de utilizarse es el de herencia (ver por ej. (Atkinson et al., 2003; Henderson-Sellers and Gonzalez-Perez, 2006)).

4.5. Resumen

Comprensión de conceptos
Modelo
Metamodelo
Lenguaje de modelización
Relaciones con la orientación a objetos
Enfoque OMG
Sobreutilización de *Class*
Sobreutilización de *Instance*
Carencia de diagramas orientados a objetos
Complejidad de modelo
Niveles de modelización
Modelo raíz
Bidireccionalidad
Cardinalidad
Metamodelización
Estereotipos
Confusión con herencia

Tabla 4.1: Resumen de los problemas de la modelización

En este capítulo hemos realizado un repaso a los principales problemas que afectan a la modelización y que pueden resumirse en los siguientes:

- La teoría de modelización más extendida, propuesta por OMG, mezcla conceptos de orientación a objetos con conceptos de modelización. Esta situación ha producido intensos debates sobre los conceptos de modelización en la literatura, poniendo de manifiesto la falta de consenso incluso en los conceptos más básicos.
- La sobreutilización de clase, instancia y herencia conduce a que la especificación de UML, el *lenguaje de modelización* más extendido, sea muy compleja, confusa y con una semántica definida de forma imprecisa.
- La propuesta de usar los *UML Profiles* para definir lenguajes específicos en un dominio determinado (DSL) partió de una concepción igualmente confusa mezclándose dos niveles de abstracción diferentes (M2 y M3). De esta forma los diseñadores de modelos se encuentran con el dilema de utilizar directamente MOF (en M3) o los *UML Profiles* (en M2 a nivel de uso y M3 a nivel conceptual).

- No hay un modelo raíz ampliamente aceptado. Aunque MOF fue propuesto para ese fin, sus imbricadas relaciones con UML a través de librerías básicas (Core) hacen difícil hacen que sea escasamente utilizado como tal.
- La obtención de unos modelos por instanciación de otros (metamodelización) se ha confundido y sigue confundiéndose con la obtención de modelos como una especialización de otros (generalización). Una vez más, los conceptos de orientación a objetos, instanciación y herencia, se mezclan y producen confusión con los conceptos de modelización.

En los capítulos siguientes abordaremos una propuesta que permita dar una respuesta a estos problemas y ayude a una compresión más clara de la modelización.

Capítulo 5

Principios sobre los fundamentos de la modelización

En el capítulo 4 se ha puesto de manifiesto que la modelización como nuevo paradigma de la ingeniería del software está lejos de ser una disciplina bien comprendida. Aunque se ha mostrado que existe mucho interés de la comunidad científica en aplicar la modelización, especialmente en el ámbito de las TIC, los estudios de uso real, tal como vimos en 2.2.3, indican que su utilización es inferior a la de otras metodologías como la orientación a objetos.

Cabe preguntarse cuáles son las causas de esta situación y qué podría hacerse para que la modelización llegase a ser en la práctica ese nuevo paradigma que permita alcanzar los tradicionales objetivos de la ingeniería del software. En nuestra opinión el factor principal que permitiría alcanzar ese objetivo es el de que la modelización esté fundamentada en unos principios teóricos sólidos y aceptados, como ya ocurrió con los anteriores paradigmas de la programación estructurada o la orientación a objetos. Ese objetivo aún está lejos de alcanzarse por diversos motivos. En primer lugar, los debates en la literatura no son sólo numerosos (algo habitual cuando aparece un nuevo paradigma) sino que se llevan produciendo en un largo periodo de tiempo (más de diez años) sin que aún se haya conseguido un consenso. En segundo lugar, no existe una propuesta de teoría básica que constituya los cimientos del nuevo paradigma. En tercer lugar, las propuestas sobre modelización han estado basadas en el anterior paradigma de la orientación a objetos y por tanto, se carece de unos fundamentos teóricos de la modelización realmente independientes y aceptados por la literatura. En cuarto lugar, no se dispone de una teoría básica comprensible, pues de serlo no se producirían los

numerosos debates sobre las dificultades de la modelización que hemos descrito. Finalmente, una teoría consensuada, con fundamentos teóricos sólidos, independiente y comprensible, tiene que ser usable para que tenga éxito en la práctica.

Por todas estas razones entendemos que tiene sentido investigar cuáles son esos principios con el fin de ofrecer una propuesta de teoría de los fundamentos de la modelización. Ése será el objetivo de este capítulo.

No se trata de una tarea fácil, tras la modelización está la propia esencia del razonamiento humano: descubrir cómo vemos la realidad para entenderla, describirla e incluso construirla a nuestro antojo. La orientación a objetos nos ha ofrecido un modo de ver esa realidad que puede aplicarse en muchas circunstancias, pero no de forma universal, y a pesar de ello hemos llegado a pensar que realmente *todo es un objeto*. La modelización también ha aspirado a ofrecer esa generalidad en su aplicación, llevándonos a pensar similarmente que *todo es un modelo* (Bézivin, 2005). Aunque ninguna de las dos afirmaciones es absolutamente cierta, ilustran cómo cualquier entidad *podría verse* como un objeto o como un modelo e incluso hacernos deducir erróneamente que todo objeto es un modelo y viceversa.

Modelo y objeto no son los únicos términos con ese propósito general. En el estudio de la modelización aparecen otros términos, también básicos, como el de *lenguaje*, *sistema* o el de *concepto* en sí mismo. Entender las relaciones entre ellos tampoco es tarea fácil y profundizar en su conocimiento conduce inexorablemente hacia otras disciplinas como la filosofía, la semiótica o la teoría de sistemas. Un ejemplo en este sentido lo constituye la propuesta conocida como *Fenomática* (Dominguez, 1999), que se basa en la consideración de principios epistemológicos, basados en la Fenomenología. Tras realizar un estudio preliminar de esta propuesta hemos llegado a la conclusión de que se trata de una teoría tan compleja que requiere de una comparativa mucho más detallada y exhaustiva, con lo que esta tarea queda planteada como trabajo futuro (ver **Conclusiones y líneas de trabajo futuro**).

La mayor parte de la complejidad asociada a la modelización procede precisamente de la dificultad para definir las relaciones entre todos estos conceptos básicos. Desafortunadamente, muchos de los autores que han analizado estas relaciones han tomado como referencia el enfoque dominante orientado a objetos de OMG, lo que lejos de clarificar ha añadido más confusión si cabe a este problema.

Desde nuestro punto de vista, la modelización sólo podrá consolidarse como

paradigma cuando se desprenda de su forzada dependencia de la orientación a objetos. De hecho, en verdad es la orientación a objetos quien depende de la modelización y no al revés, al tratarse de una teoría más general en la que, como veremos, está la esencia del pensamiento y más concretamente la del pensamiento científico.

A modo de resumen, en nuestra opinión una verdadera teoría de la modelización como mínimo debe:

- Centrarse en el concepto de modelo y exponer con claridad qué es, cómo se crea y cómo se relaciona con las entidades que modeliza
- Explicar con claridad las relaciones entre modelo y otros conceptos básicos como lenguaje o sistema
- Clarificar las diferencias y similitudes con el anterior paradigma de la orientación a objetos
- Permitir explicar el origen de los problemas y confusiones debatidos por la literatura durante años

Este capítulo describirá los tres principios básicos en los que se basa nuestra propuesta hacia una teoría sobre los fundamentos de la modelización, que cumpla estos requisitos y que será descrita en los capítulos siguientes.

El primer principio es el de *generalidad*. Consiste en intentar que los fundamentos de la modelización sean aplicables en el mayor número de ámbitos posible, no sólo el de las TIC o la ingeniería del software. La aplicación de este principio no es en absoluto sencilla. Es difícil encontrar en la literatura propuestas de teoría de la modelización con esa visión general. Efectivamente, la mayoría de la literatura sobre modelización revisada se encuadra dentro del ámbito de las tecnologías de la información y especialmente en el contexto de los debates sobre OMG, como UML o MOF. Sin embargo, la modelización es de aplicación a muchas áreas de conocimiento. Así por ejemplo existen modelos de negocio, económicos, arquitectónicos, de aprendizaje, etc. Aunque el origen de nuestra propuesta se sitúa en la investigación de los modelos conceptuales de seguridad en SOA descrita en el capítulo 2, nuestro objetivo al aplicar este principio es el de proponer unos fundamentos de la modelización que puedan ser de aplicación para la comprensión y uso de cualquier tipo de modelo y no sólo un ámbito muy concreto, como por ejemplo, en las TIC o conforme a los estándares OMG.

El segundo principio es el de *independencia*. Implica situar en el centro de nuestro estudio el concepto de modelo, independizando su definición de otros con-

ceptos como lenguaje o sistema, que como hemos visto aparecen habitualmente en las definiciones. La aplicación de este principio nos llevará a buscar respuesta a alguna de las preguntas ya planteadas, tales como si puede existir un modelo sin lenguaje o si pueden existir modelos de entidades que no son sistemas. Como describiremos con más detalle más adelante, el conocido como *triángulo del significado* (*meaning triangle*) servirá como referencia para su comprensión y aplicación.

Finalmente, el tercer principio es el del *enfoque científico*. Este principio es esencial para la comprensión de nuestra propuesta y su aplicación facilita la de los otros principios. Se trata de abordar el estudio de la modelización desde un nuevo punto de vista: el del método científico cuya generalidad es bien conocida (es aplicable a multitud de ámbitos). Igualmente sus fundamentos no están condicionados por otros conceptos facilitándonos la aplicación del principio de independencia. En el apartado 5.2 describiremos en profundidad las características principales del método científico que ayudarán a comprender su aplicación en nuestro enfoque de la modelización.

La propuesta que presentaremos en los siguientes capítulos no rompe radicalmente con los puntos de vista de la mayoría de los autores respecto a qué es un modelo. De hecho, como se verá más adelante, su definición de modelo se basa en una de las definiciones más aceptadas. Sin embargo, la aplicación simultánea de estos tres principios sí supone un cambio de enfoque que ofrece una nueva forma de abordar el problema de los fundamentos de la modelización.

5.1. Principio de independencia. Triángulo de la modelización

De la revisión de las definiciones de modelización realizada en el capítulo 4 pudimos comprobar que existen dos conceptos, sistema y lenguaje, tratados habitualmente de forma conjunta con el de modelo desde distintos puntos de vista y a veces usando diferentes terminologías. Estos conceptos no sólo son el centro de muchos de los debates sobre modelización encontrados en la literatura sino que muchos autores también los incluyen en la propia definición de modelo, tal como se puede observar en su RCM.

El origen de la asociación entre estos tres conceptos tiene relación con el conocido como *triángulo del significado* (*meaning triangle*) o *triángulo semiótico*

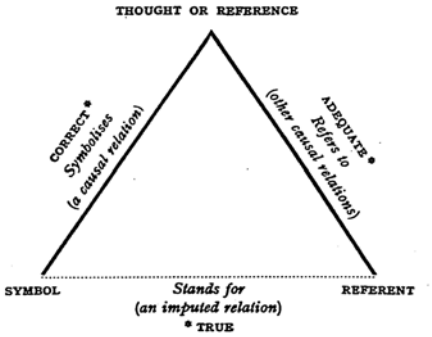
(*semiotic triangle*), cuyos vértices se corresponden con los siguientes elementos:

- **Objeto real**: representa a la realidad y también es conocido como *referente* o simplemente *cosa*.
- **Concepto**: los objetos reales son percibidos por un observador quien mediante un proceso de abstracción obtiene un concepto de dicho objeto. Percepción y Conceptualización son procesos mentales subjetivos.
- **Símbolo**: es una representación sensorialmente perceptible. No existe un camino directo entre símbolo y referente sino que el significado de un símbolo proviene del concepto al que se refiere.

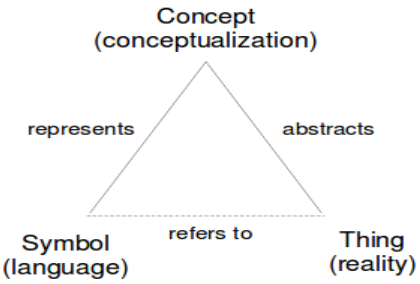
Este triángulo también ha recibido el nombre de algunos de los autores que lo han analizado en el contexto del estudio del significado y de la semiótica, y que propusieron versiones del mismo tales como el *triángulo de Ullman* o el *triángulo de Ogden-Richards*. En la figura 5.1 se muestran algunos de los más relevantes.

Una variante del triángulo del significado es el que (Gupta and Sykes, 2001) denominaron *triángulo de modelo* (*model triangle*). Posteriormente, uno de los autores (Sykes, 2003), realizó una nueva versión que denominó *triángulo de la modelización* (*modeling triangle*). Según esta propuesta, Sykes además de cambiar el orden de *symbolic system*, cambia el nombre de *concrete system* por *perceived system*. Asímismo indica que el *triángulo de la modelización* es una forma de triángulo del significado en la que los elementos individuales (símbolo, objeto y concepto) son sistemas, es decir, sistemas de símbolos (lenguaje), sistema de objetos (que el autor denomina sistema percibido) y sistema de conceptos. El autor centra su atención en los sistemas e incluso va más allá identificando modelos con sistemas. Otro de los aspectos interesantes de esta propuesta es que a partir del *triángulo de la modelización*, Sykes define la creación e interpretación de modelos. En efecto, como puede observarse en la figura 5.2, la creación de un modelo sería el resultado de un proceso de conceptualización (paso de sistema percibido a sistema conceptual) seguido de una representación (de sistema conceptual a sistema de símbolos), mientras que la interpretación de un modelo es una realización seguida de una aplicación. Las líneas de puntos entre los nodos *perceived system* y *symbolic system* indican que no hay un camino directo entre los sistemas simbólico y percibido (tal como también ocurre en algunas versiones del *triángulo del significado*).

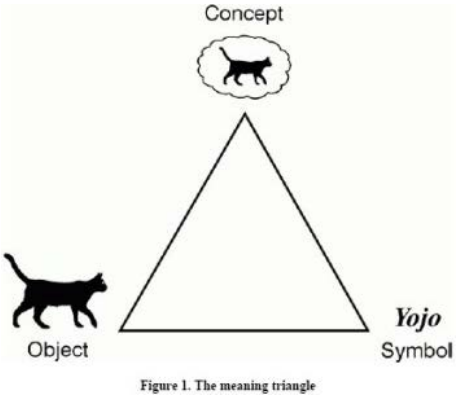
En la misma línea Noyrit et al. (2010), distingue tres niveles (layers): nivel de la realidad, nivel de la conceptualización y nivel del lenguaje. A su vez en el



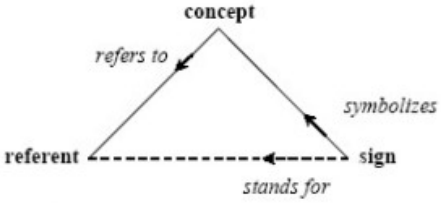
(a) (Ogden and Richards, 1923)



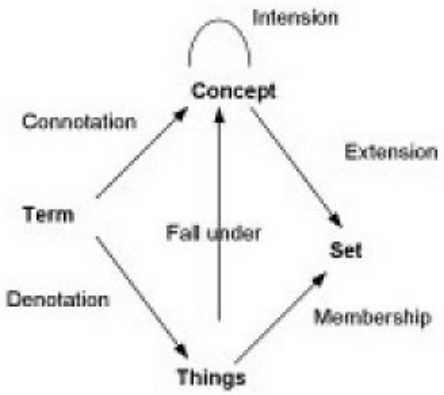
(b) (Ullmann, 1962)



(c) (Sowa, 2000)



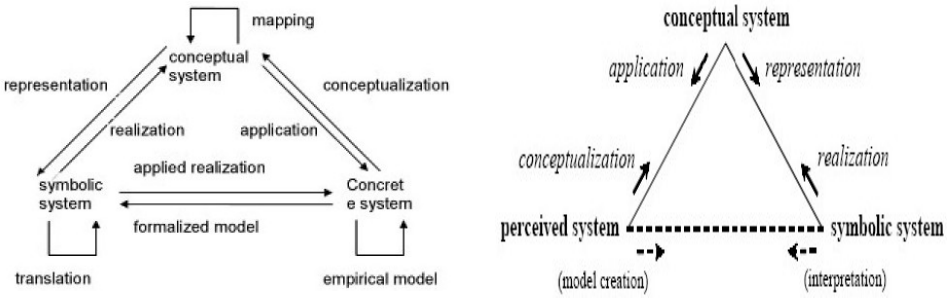
(d) (Sykes, 2003)



(e) (Palomaki, 1994)

Figura 5.1: Triángulos del significado

último nivel diferencia entre sintaxis concreta y abstracta. Pasando al plano de la semiótica, los autores asocian respectivamente sintaxis, semántica y pragmática a los niveles de lenguaje, conceptualización y realidad, concluyendo que desde el punto de vista de la sintaxis:



(a) Model Triangle (Gupta and Sykes, 2001)

(b) Modeling Triangle (Sykes, 2003)

Figura 5.2: Triángulos de la modelización

> It defines the rules for constructing well-formed language statements (models) using a set of symbols

Las relaciones entre la semiótica y la modelización propuestas por estos autores confirman nuestra afirmación sobre las dependencias entre lenguaje, modelo y sistema, sin embargo mantienen abiertos interrogantes respecto al significado de modelo respecto a sistema y lenguaje. Por ejemplo, podemos preguntarnos si es posible modelizar algo que no sea un sistema o si un modelo puede existir sin que sea expresado en un lenguaje de modelización. Si la respuesta a estas preguntas fuese afirmativa, la definición de modelo como una descripción de un sistema expresada en un lenguaje tendría que ser revisada. A la inversa pueden plantearse otras preguntas interesantes tales como si todo sistema es un modelo o si un lenguaje puede ser un sistema o un modelo.

En cualquiera de los casos podrán obtenerse respuestas si nos adentramos en el significado de estos dos conceptos. Éste ha sido nuestro enfoque: primero abordaremos el significado de modelo de la forma más independiente posible de los conceptos de sistema y lenguaje. Posteriormente, trataremos el significado de sistema y lenguaje desde el punto de vista de la modelización. Podría decirse que se ha seguido un camino de algún modo inverso al de buena parte de la literatura puesto que en lugar de definir modelo a partir de sistema y lenguaje, definiremos sistema y lenguaje a partir de modelo, poniendo de manifiesto la importancia de la teoría de la modelización para la comprensión de éstos y otros conceptos relacionados.

La figura 5.3 muestra la versión inicial de nuestra propuesta de triángulo de la modelización, también basado en el triángulo del significado, indicando sólo

sus vértices. Por el momento sólo resaltaremos que en primer lugar el triángulo no es tal, pues le falta un lado (la unión entre original y expresión). En segundo lugar, obsérvese que el vértice *expresión* está en gris, indicando que podría no existir, o lo que es lo mismo, que un modelo puede existir sin ser expresado. Profundizaremos en esta cuestión en los capítulos siguientes.

- **Modelo**: a diferencia de los otros triángulos, aparece Modelo como uno de los vértices en el lugar de concepto o sistema conceptual, lo cual es consistente con el resto de triángulos puesto que un concepto es una representación mental y desde nuestro punto de vista los modelos son entidades abstractas que habitualmente residen en la mente de un modelizador.
- **Original**: aquello que es modelizado. Aunque puede ser un sistema, en general será cualquier entidad. Como veremos más adelante, coincidimos con (Gupta and Sykes, 2001) en que no se refiere a entidades físicas sino a percibidas.
- **Expresión**: aunque puede ser un sólo símbolo, indicamos en general una expresión de algún lenguaje

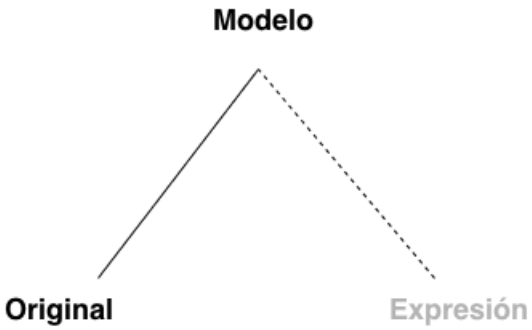


Figura 5.3: Versión inicial de nuestra propuesta de Triángulo de la modelización

En capítulos posteriores iremos describiendo las relaciones (lados del triángulo) y las implicaciones de este enfoque.

5.2. Principio del enfoque científico. Definición y fases.

En realidad no existe *un* método científico, entendido como un único conjunto de reglas que siguen de manera sistemática todos los científicos en su labor de

investigación. La verdadera naturaleza de las técnicas utilizadas por los científicos ha sido objeto de un intenso debate a lo largo de los siglos. Este debate se encuadra entre lo que se conoce como *filosofía de la ciencia* cuyo fin último es la investigación del conocimiento científico desde un punto de vista filosófico. Personajes tan relevantes como Aristóteles, Galileo, Francis Bacon, Descartes, Isaac Newton, Leibniz, Bernoulli, Hume, Thomas Bayes, John Stuart Mill, Henri Poincaré, Laplace, Charles S. Peirce, John M. Keynes, Karl Popper, Rudolf Carnap o Thomas Kuhn entre muchos otros, aportaron diferentes visiones sobre cómo el ser humano adquiere el conocimiento científico (Gower, 2002).

Según el Collins English Dictionary el método científico se define como

> A method of investigation in which a problem is first identified and observations, experiments, or other relevant data are then used to construct or test hypotheses that purport to solve it

Sin entrar en la infinidad de matices que conlleva la discusión acerca del método científico, a continuación describiremos los principales pasos que definen el método científico, y que hemos representado de manera resumida en la figura 5.4.

5.2.1. Formulación del problema y finalidad

La adquisición de conocimiento científico siempre parte de la formulación de preguntas sobre un problema concreto, generalmente para conseguir una finalidad determinada mediante la investigación.

5.2.2. Hipótesis

Las hipótesis constituyen el centro del método científico. Una hipótesis es una suposición que se establece sobre los hechos relacionados con el problema formulado y sobre la que aún ha de ser comprobada su veracidad.

La principal cuestión que surge en este punto es cómo se obtienen las hipótesis a partir de la formulación del problema. A lo largo de los siglos ha habido encendidos debates entre los partidarios de diferentes enfoques. A continuación describiremos dos de los principales métodos de obtención de las hipótesis: el método analítico y el sintético.

Método analítico

Analizar consiste en descomponer un todo en cada una de sus partes para estudiarlas de manera separada identificando las relaciones que existen entre sí y entre éstas y el todo. La aplicación del análisis para la obtención de hipótesis conlleva una serie de pasos:

1. **Observación**: consiste en la recepción y recopilación de información sobre uno o más hechos en el *ámbito de estudio*, a través de los sentidos, bien directamente o por la captura de datos mediante instrumentos. La finalidad delimita el *ámbito de estudio*. Puesto que no es posible abarcar la totalidad de la realidad es imprescindible concretar cuál es el ámbito del estudio científico que se pretende llevar a cabo.

   Podemos distinguir dos modos de observación:

   - **Directa**: se observan los hechos relacionados con el *ámbito de estudio* directamente
   - **Indirecta o por analogía**: es muy difícil o imposible la observación directa y en su lugar se observan hechos de otro *ámbito de estudio* similar o análogo.

2. **Abstracción**: es el proceso mental consistente en distinguir aquellas características esenciales de lo percibido que estén relacionadas con el *ámbito de estudio*. La abstracción es importante en un proceso analítico porque permite diferenciar las partes de un todo separando lo relevante de lo que es superfluo respecto al problema planteado.

3. **Inducción**: es un tipo de inferencia donde las premisas son los hechos observados y las conclusiones son proposiciones que se postula que son ciertas de manera general para dichos hechos en el *ámbito de estudio*. Estas proposiciones son las hipótesis. Desde un punto de vista analítico las hipótesis permiten expresar en forma de proposiciones las supuestas relaciones que existen entre las partes de lo observado que han sido obtenidas a través de un proceso de abstracción.

   Existen dos tipos de inducción:

   - **Inducción completa**: cuando las hipótesis son obtenidas por la observación de todos los hechos dentro *ámbito de estudio*.
   - **Inducción incompleta**: no es posible estudiar todos los hechos en el *ámbito de estudio* y por tanto es necesario elegir un subconjunto de

ellos o muestra representativa a partir de la cual formular las hipótesis.

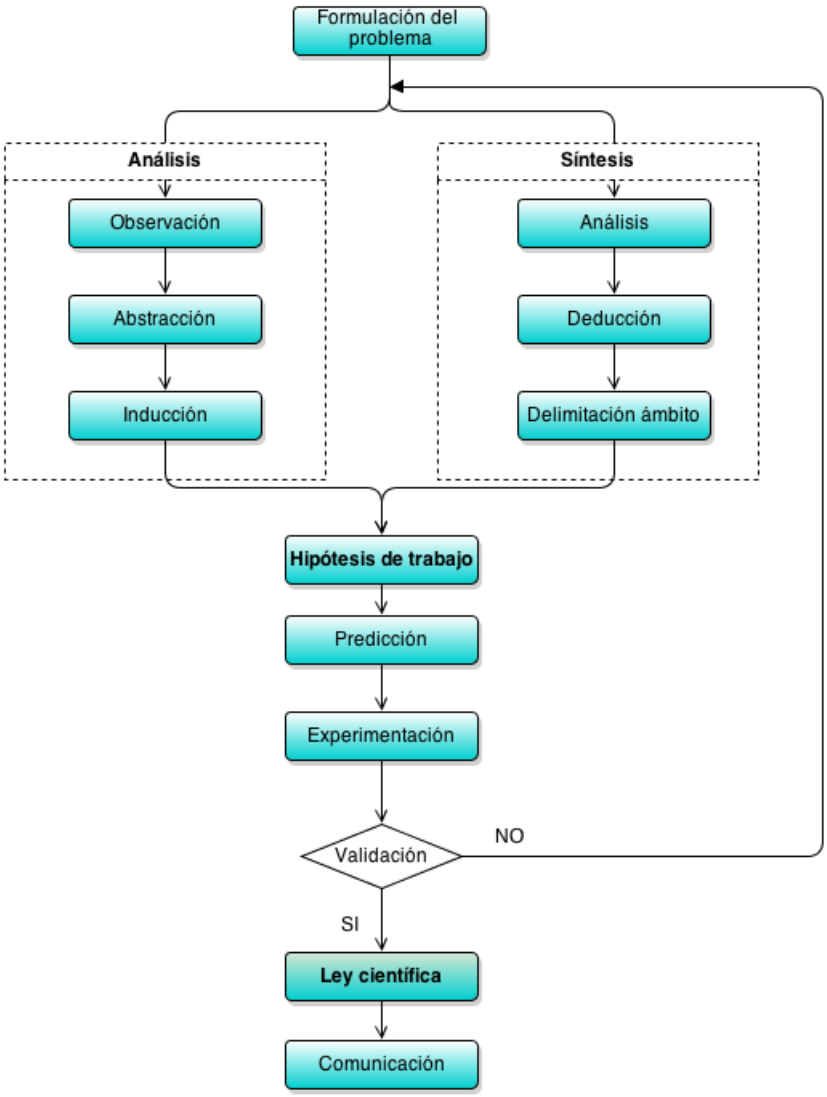


Figura 5.4: Fases del método científico

Método sintético

Sintetizar es de algún modo la operación inversa al análisis. Significa volver a construir e integrar las partes de un todo, pero no para obtenerlo de nuevo, sino para comprender mejor las relaciones entre sus partes, y entre éstas y el todo, desde una nueva perspectiva.

La aplicación de la síntesis para la obtención de hipótesis es un modo muy diferente al del análisis puesto que a diferencia de éste usa el razonamiento sin

tomar como punto de partida la observación de la realidad. Igualmente, el método sintético conlleva una serie de pasos:

1. **Análisis**: de la definición de síntesis se deduce que las partes del todo que se quiere sintetizar han debido obtenerse previamente mediante análisis. Por tanto, análisis y síntesis son en realidad procesos complementarios. A diferencia del método analítico, en el método sintético el análisis tiene como objeto no hechos observados del *ámbito de estudio*, sino principios o proposiciones existentes cuyo análisis permitirá, en un plano conceptual, identificar aquellos aspectos a partir de los cuales obtener las hipótesis. Estas proposiciones existentes pueden ser de diferente naturaleza. Desde un punto de vista matemático, si son consideradas verdaderas pero su veracidad no es demostrada, se denominarían axiomas, mientras que si se trata de proposiciones que se suponen ciertas, pero cuya veracidad no ha sido probada ni refutada hasta el momento, estaríamos hablando de conjeturas.
2. **Deducción**: una vez realizado el análisis se trata de obtener las hipótesis a partir de las proposiciones analizadas. El método más utilizado es el de la deducción, el cual al igual que la inducción es un tipo de inferencia. Sin embargo, en este caso las premisas son las proposiciones sobre las que se ha realizado un análisis y las conclusiones son las nuevas proposiciones obtenidas a partir de las anteriores y que se postulan como hipótesis.
3. **Delimitación del ámbito de estudio**: el siguiente paso es obtener el ámbito de estudio donde se postula que las hipótesis serán verdaderas. Aunque, desde un punto de vista matemático, unas proposiciones obtenidas mediante deducción sean ciertas, podrían no serlo en el *ámbito de estudio* donde se ha formulado el problema. Obsérvese que es la situación inversa a la del método analítico, donde el primer paso es la observación de la realidad para delimitar el *ámbito de estudio*.

   Si se parte de hipótesis ya contrastadas en otro ámbito de estudio, pero se deduce que son aplicables de manera total o parcial al *ámbito de estudio* de la investigación, estaríamos nuevamente en un caso de analogía.

5.2.3. Predicción

Una vez formulado el problema y unas hipótesis para su resolución en un *ámbito de estudio* determinado, estaremos en condiciones de enunciar unas predicciones

basadas en las hipótesis. La predicción consiste en adivinar la ocurrencia de hechos dentro del *ámbito de estudio*.

Las predicciones son resultado de un proceso sintético en el que el punto de partida ahora son las hipótesis. A partir de ellas se intentan obtener consecuencias desconocidas o imprevistas mediante deducción. Las predicciones tienen que estar referidas a hechos pasados, presentes o futuros dentro del *ámbito de estudio*. Mediante la validación de las previsiones pueden sacarse conclusiones sobre la validez de las hipótesis a partir de las cuáles se han obtenido.

Hay dos factores que dan credibilidad a priori a una predicción:

- Imprevisión: si las hipótesis han sido obtenidas mediante análisis, los hechos observados de las que parten no son lógicamente sorprendentes. Una hipótesis adquiere más valor si es capaz de predecir hechos futuros.
- Exclusividad: las hipótesis deben ser capaces de predecir y explicar hechos que no pueden ser explicados por otras hipótesis. Por ejemplo, tanto la hipótesis de que el Sol da vueltas alrededor de la Tierra (teoría geocéntrica), como la hipótesis de que la Tierra da vueltas alrededor del Sol (teoría heliocéntrica), pueden explicar el día y la noche. Sin embargo, sólo la teoría heliocéntrica podía explicar otros hechos, como por ejemplo los movimientos de Marte.

5.2.4. Validación y experimentación

La validación de una predicción obtenida a partir de una hipótesis es uno de los problemas más complejos y en el que no ha habido acuerdo por parte de filósofos y científicos a lo largo de los siglos.

Validar consiste en comprobar que la predicción se corresponde con la realidad, es decir, es conforme a los hechos observados. Para validar es necesario observar nuevamente la realidad y en la mayoría de los casos volver a recopilar datos guiados por las características relevantes presentes en las hipótesis y que han sido obtenidas mediante un proceso de análisis o de síntesis.

Existen varias cuestiones asociadas a dicha comprobación:

- Experimentación: no siempre es posible obtener de manera espontánea los datos necesarios para validar una predicción sino que es necesario diseñar experimentos específicos para aislar aquellas características que se pretenden medir para comprobar las predicciones. Sin embargo, la experimentación,

aún siendo importante, no es una condición necesaria para que una investigación pueda ser denominada como científica puesto que en algunas áreas de conocimiento como la vulcanología o la astronomía es difícil o imposible realizar experimentos. En estos casos la investigación ha de basarse exclusivamente en la observación de los hechos producidos de manera natural y no controlada.

- Reproducibilidad: la experimentación científica implica una de las características relevantes del método científico, la reproducibilidad de los hechos. Si un hecho no puede ser reproducido por cualquier investigador en las mismas condiciones que el investigador original, entonces la validación de la teoría pierde valor. A pesar de esto, tampoco es una característica necesaria para que una investigación pueda ser considerada como científica puesto que existen experimentos muy difíciles de reproducir en las mismas condiciones por su naturaleza (por ejemplo, en las ciencias humanas y sociales) o su alto coste.
- Comprobación directa: tal como hemos indicado anteriormente al referirnos a la observación de los hechos, en ocasiones es difícil o imposible observar la realidad en el *ámbito de estudio* y por tanto se torna muy complicado obtener la información necesaria para la comprobación de las predicciones y por tanto la veracidad de las hipótesis. En estos casos, generalmente se recurre nuevamente a la analogía: de hechos análogos se deduce si una predicción se cumpliría para los hechos del *ámbito de estudio*.
- Certeza probable: supongamos que las predicciones se cumplen para todos los hechos asociados al *ámbito de estudio*. En ese caso podríamos afirmar que las hipótesis son ciertas. Sin embargo, en la práctica esto no suele ser posible y sólo podremos estar seguros en un número no completo de los hechos. Cabe preguntarse en qué grado podríamos estar seguros de la veracidad de las hipótesis. Históricamente el debate versó sobre si una mayor cantidad de predicciones verificadas hacen que una hipótesis sea cierta o *más* cierta.
- Falsabilidad y corroboración: Karl Popper propuso un enfoque completamente diferente (Gower, 2002). En lugar de centrarse en la veracidad de las predicciones planteó que sólo podría verificarse su falsedad. Para Popper una hipótesis no refutada (es decir, todas las predicciones que se han comprobado son ciertas) no puede decirse que sea cierta sino que ha sido *corroborada*.

El método científico es un proceso iterativo. Si las hipótesis no son verificadas o corroboradas el investigador deberá revisarlas comenzando de nuevo el proceso utilizando el método analítico o sintético y empleando sus capacidades de inducción o deducción.

5.2.5. Comunicación

El último paso es uno de los más importantes puesto que nada de lo anterior tendría valor si el investigador se lo guarda para sí mismo.

El científico comunica su trabajo al resto del mundo publicándolo de algún modo, para que cualquiera pueda repetir sus pasos y refrendar los resultados. Una vez aceptado por la comunidad, sus hipótesis verificadas pasan a ser leyes científicas. Un conjunto de leyes relacionadas conforman una teoría científica. A lo largo de la historia el progreso científico ha sido posible gracias a que nuevas teorías han surgido basándose o corrigiendo a las anteriores en un continuo acercamiento a la verdad científica y al conocimiento.

Para realizar la comunicación de su trabajo, los científicos utilizan un lenguaje, (o más bien un conjunto de lenguajes científicos), que en muchos casos sólo es comprensible por los investigadores de las mismas áreas de conocimiento debido principalmente al uso de una terminología específica.

Capítulo 6

Fundamentos de la modelización

En este capítulo abordaremos las principales definiciones de nuestra teoría sobre los fundamentos de la modelización. Estas definiciones están basadas en los principios descritos en el capítulo anterior, constituyen el núcleo de la teoría y tratan sobre los conceptos principales de la modelización centrados en el concepto de modelo. En los capítulos siguientes se definirán otros conceptos de modelización, también importantes pero cuya comprensión está basada en los conceptos principales descritos en este capítulo.

Lógicamente el primer concepto que ha de abordarse en cualquier teoría de modelización es el de modelo. Hemos visto de la revisión de la literatura, del análisis de los problemas de la modelización y sobre todo del RCM de modelo que su definición se presta a numerosas interpretaciones incluso contradictorias. Nuestro objetivo no es introducir aquí una definición de modelo que se separe completamente de las definiciones propuestas en la literatura sino proponer una definición conforme a los principios de generalidad, independencia y enfoque científico descritos en el capítulo anterior.

Para conseguir nuestro objetivo elegimos como punto de partida una definición que ofrece un punto de vista interesante, propuesta por Seidewitz (2003):

> A model is a set of statements about some system under study (SUS)

y que posteriormente fue reformulada por Gonzalez-Perez and Henderson-Sellers (2007):

> A model is a statement about a given subject under study (SUS), expressed in a given language

Estas dos definiciones son lo suficientemente genéricas para que puedan ser aplicadas a cualquier ámbito siguiendo el principio de generalidad descrito en el capítulo

5. Por otra parte, como veremos más adelante, son definiciones que sirven de base para la aplicación del principio del enfoque científico.

Sin embargo, también tienen algunas limitaciones:

- No son definiciones esenciales según las métricas del RCM de modelo. Efectivamente, como vimos en el estudio de las métricas del RCM de Modelo (ver sección 4.1.1) las definiciones esenciales de modelo eran las que hacían referencia a los conceptos de descripción, especificación y abstracción.
- Carecen de otros aspectos que consideramos igualmente importantes en la noción de modelo relacionados con su creación, interpretación, uso y utilidad y que están relacionados con el método científico descrito en la sección 5.2.
- Las definiciones se apoyan respectivamente en los conceptos de sistema y lenguaje mientras que según describimos en la sección 5.1 nuestro enfoque, según el principio de independencia, es considerar modelo como un concepto primario independiente de sistema y lenguaje definiendo estos conceptos a partir de modelo pero no a la inversa.

Estas definiciones, como otras encontradas en la literatura, en nuestra opinión abordan un concepto previo que es la base de un modelo pero que no es realmente un modelo y que denominaremos *protomodelo*.

6.1. Protomodelos

Las definiciones de Seidewitz y Gonzalez-Perez and Henderson-Sellers son aparentemente sencillas. Sin embargo, si intentamos aplicarlas deberemos entender el significado de algunos de los conceptos que aparecen directa o indirectamente en esas definiciones. Veámoslo con un ejemplo. Supongamos que queremos expresar el modelo de mesa según el punto de vista de estos autores. Para ello deberíamos obtener sentencias sobre mesas tales como las siguientes:

- Tiene una parte superior.
- Tiene una o más patas situadas bajo la parte superior
- Sus patas sostienen a su parte superior.

La primera cuestión con la que nos encontramos es la de entender el concepto de sentencia. Tendremos que tener claro qué es una sentencia, cómo se diferencia este término de otros aparentemente sinónimos como proposición, si nos sirve cualquier tipo de sentencia o cómo debería ser expresada, en su caso, para que pudiese

formar parte de un modelo. Estas cuestiones ya presentan algunas dificultades y pospondremos su explicación para los siguientes apartados.

En segundo lugar se entiende que para aplicar un modelo las sentencias deberían ser evaluadas por algún sujeto que llamaremos modelizador. Esta cuestión nos lleva a un aspecto importante en los modelos, la pragmática, que será tratado con más profundidad en el capítulo 9. Por el momento para nuestro ejemplo nos bastará con tener en cuenta que la evaluación de las sentencias la realiza el modelizador y por tanto será éste quien podrá determinar si la sentencia es cierta o falsa para las entidades que queremos conocer si son o no mesas. Sin embargo, esa evaluación tampoco es sencilla, puesto que para poder evaluar hay que observar y eso nos lleva al concepto de características de una entidad, que son las cualidades identificadas por el modelizador, para primero construir sus sentencias y después evaluarlas en las entidades observadas. Se trata de un concepto que ha sido debatido filosóficamente cuya comprensión es esencial para entender ese proceso de evaluación y que dada su complejidad intentaremos describir con más detalle en los siguientes apartados.

La siguiente cuestión es determinar qué sentencias forman un modelo según este enfoque. Por ejemplo, las sentencias anteriores podrían ser evaluadas como ciertas para un gato y por tanto habría que refinarlas para asegurarnos que las entidades que se están modelizando son realmente mesas, añadiendo otras sentencias como las siguientes:

- Es un ser inerte.
- La parte superior es una superficie plana.

Con esas sentencias podríamos pensar que ya tenemos el modelo de mesa según estos autores. Sin embargo, nos surge otra cuestión relevante. Una mesa es una entidad física y como tal no podemos evaluarla en una sentencia que es algo no material. Parece entonces que tanto el modelo como la entidad que queremos modelizar han de llevarse a un plano mental, o dicho de otro modo, conceptual. Si profundizamos un poco en esta cuestión veremos que el propio concepto de *concepto* nos lleva nuevamente al plano filosófico y está muy relacionada con el lenguaje. Se trata de un problema complejo cuya explicación pospondremos igualmente a uno de los siguientes apartados.

Otros términos importantes que aparecen en estas definiciones son los de sistema y lenguaje de modelización. Como ya indicamos en el capítulo 5 los abordaremos más adelante desde nuestra óptica de la modelización, aplicando el Principio

de independencia, una vez presentados los fundamentos de nuestra propuesta.

La última, aunque no menos importante cuestión que podemos plantearnos con este ejemplo es ¿hemos obtenido realmente el modelo de mesa una vez que tenemos identificado el conjunto de sentencias que son ciertas para todas las mesas? En nuestra opinión, ese conjunto de sentencias forma parte de un modelo pero es insuficiente porque sólo nos permite validar las entidades que se quieren modelizar. Pero un modelo es algo más que eso, tal como explicaremos en este capítulo.

Por esa razón, y para diferenciarlo de nuestra concepción de modelo, hemos denominado *protomodelo* a este enfoque preliminar e incompleto de lo que es un modelo. Obsérvese que las cuestiones que hemos introducido con este ejemplo de sentencias también aparecen con otros ejemplos más habituales en la literatura sobre modelización. Por ejemplo un diagrama de clases. En efecto, ese tipo de diagramas constituye un conjunto de sentencias expresadas gráficamente y que podrían permitir realizar alguna validación referida a algunas entidades. En ese caso, desde nuestra perspectiva ese diagrama, o más precisamente, el conjunto de sentencias expresadas mediante ese diagrama, también es en realidad un protomodelo.

Dada la importancia de *protomodelo* como punto de partida para nuestra propuesta, lo definiremos a continuación de la forma más precisa posible junto con otros conceptos asociados.

6.1.1. Protomodelo

En esencia un protomodelo es una entidad asociada a un conjunto de sentencias y cuya certeza sobre otra entidad llamada original puede ser obtenida a través de una función de validación construida a partir de dichas sentencias. La función de validación y en consecuencia sus sentencias, es el elemento principal que relaciona un protomodelo de alguna entidad con dicha entidad. Más formalmente podemos expresarlo a través de las siguientes definiciones:

Sea *entidad* todo aquello que tenga una existencia (no necesariamente material) separada y diferenciada de otras entidades, siendo $\mathcal{E}$ el conjunto de todas las entidades que denominaremos *Universo de entidades*.

Definición **6.1.** Sentencia

Se define **sentencia** como una afirmación d sobre entidades de forma que, dada una entidad particular e, pueda decidirse si e hace a la afirmación verdadera

o falsa, es decir, $e \in \mathcal{E}, d : \mathcal{E} \rightarrow \mathcal{B}$ siendo $\mathcal{B} = \{true, false\}$

Definición **6.2.** Protomodelo
Se define **protomodelo** como una entidad $\mathcal{P}_m$ asociada a una **función de validación** $\mathcal{V}_{\mathcal{P}m}$ tal que considerando un conjunto de sentencias $\mathcal{D}_{\mathcal{P}m}$ es definida como $\mathcal{V}_{\mathcal{P}m} : \mathcal{E} \rightarrow \mathcal{B} | \mathcal{V}_{\mathcal{P}m}(e) = \bigwedge_{d \in \mathcal{D}_{\mathcal{P}m}} d(e)$. Se denominan **sentencias del protomodelo** a las sentencias pertenecientes al conjunto $\mathcal{D}_{\mathcal{P}m}$.

Definición **6.3.** Original
Dado un protomodelo $\mathcal{P}_m$, se define un **original** o de $\mathcal{P}_m$ como una entidad cuya evaluación de la función de validación de $\mathcal{P}_m$ para o es verdadera, es decir, $o \in \mathcal{E} | \mathcal{V}_{\mathcal{P}m}(o) = true$.

Definición **6.4.** Dominio
Dado un protomodelo $\mathcal{P}_m$, se define su **dominio** $\mathcal{O}_{\mathcal{P}m}$ como el conjunto formado por todos sus originales, es decir, $\mathcal{O}_{\mathcal{P}m} = \{o \in \mathcal{E} | \mathcal{V}_{\mathcal{P}m}(o) = true\}$

Seguidamente, teniendo presente la complejidad y el origen filosófico de las cuestiones planteadas, intentaremos dar algunas respuestas que permitan comprender mejor estas definiciones.

6.1.2. Entidades, objetos abstractos y conceptos

Como indicamos a propósito del modelo de mesa, (proto)modelo y entidad modelizada (original) se crean y evalúan respectivamente en la mente de un modelizador y son por tanto entidades no materiales. El término *modelo* usado en el lenguaje natural puede dar lugar a confusiones si no se tiene en cuenta esta matización. En efecto, podemos decir coloquialmente que una maqueta realizada en madera es el *modelo* de un edificio, pero tal entidad física no puede ser en ningún momento un *protomodelo* tal como se ha indicado en la Definición 6.2.

También podría existir cierta confusión entre protomodelos y conceptos. ¿Todo concepto es un protomodelo? ¿Todo protomodelo es un concepto? Para poder responder a estas cuestiones debe definirse previamente qué es un concepto. Lamentablemente no existe una definición universalmente aceptada de lo que és un concepto sino que ha habido diferentes interpretaciones que básicamente pueden resumirse en tres enfoques (Zalta et al., 2011):

1. **Representaciones mentales**: concepto es una representación mental utilizada por las personas para el proceso del pensamiento de una forma similar a cómo las palabras se utilizan para realizar expresiones en un lenguaje.

2. **Habilidades cognitivas**: concepto es una habilidad exclusiva de agentes cognitivos, es decir de sujetos con capacidad para conocer. Por ejemplo, el concepto de *gato*, equivale a la capacidad para discriminar los gatos de los no gatos y extraer determinadas inferencias sobre gatos.
3. **Objetos abstractos**: concepto es una entidad abstracta externa a nuestra mente. Por tanto, desde este punto de vista, puede existir un mismo concepto que tenga representaciones mentales diferentes en diferentes individuos y también pueden existir conceptos que nunca podrán ser adquiridos por la mente humana.

El debate sobre el significado de concepto está relacionado con el hecho de que un protomodelo tiene que estar realizado por alguien que llamaremos *modelizador* (*modeler*) y que de alguna forma usa los conceptos como parte de un proceso cognitivo. Obsérvese que en la segunda y tercera acepción se abre la puerta a que el modelizador podría no ser humano. En el primer caso los conceptos son siempre un producto de la mente humana muy relacionado con el lenguaje.

Sea de una forma u otra, podemos decir que en el contexto de esta teoría, un protomodelo es un tipo especial de concepto. Efectivamente, aunque no todos los conceptos son protomodelos, un protomodelo sí es un concepto puesto que, en nuestra opinión, las tres definiciones no son incompatibles entre sí. De hecho, podríamos integrar las tres definiciones considerando que un concepto es *un objeto abstracto producto de una habilidad cognitiva y que en el caso de los humanos se materializa en una representación mental.*

Por otro lado, un original también es un objeto abstracto, puesto que un objeto material no puede lógicamente ser evaluado directamente en una sentencia, que el modelizador ha creado en su mente. Un original puede estar relacionado con un objeto material pero no siempre es así (por ejemplo, como en el caso de los metamodelos, donde el original es un modelo, y que serán estudiados en el capítulo 7).

Además del propio significado de concepto otra cuestión que ha sido debatida es si pueden existir conceptos sin lenguaje y cual es la precedencia entre ambos (si la capacidad para crear conceptos aparece antes o después del aprendizaje de un lenguaje) (Zalta et al., 2011, Cap. 4). Sobre la primera cuestión, aún no habiendo un absoluto consenso, se han encontrado evidencias de que existen conceptos que podríamos llamar no-lingüísticos (que son creados sin la intervención del lenguaje). Respecto a la segunda cuestión parece razonable pensar que los conceptos

aparecen antes que el lenguaje pero por otro lado hay algunos datos que sugieren que algunos conceptos se apoyan en el lenguaje.

Nuestro punto de vista coincide plenamente con quienes reconociendo que el lenguaje ejerce una influencia en el pensamiento, y por tanto en la creación y utilización de conceptos, afirman que también pueden existir conceptos sin lenguaje (Bloom and Keil, 2001). Consecuentemente al ser los protomodelos (y modelos) un tipo especial de conceptos también podemos afirmar que existen protomodelos sin lenguaje, en línea con el principio de independencia descrito en la sección 5.1.

6.1.3. Oraciones, sentencias y proposiciones

Puede resultar extraño afirmar que pueda existir un protomodelo sin ser expresado e incluso parece una contradicción ya que la definición de protomodelo hace referencia al enunciado de sentencias y parece que éstas no puedan elaborarse sin un lenguaje. Puesto que el concepto de sentencia es importante en la definición de protomodelo, es conveniente realizar algunas aclaraciones que además ayuden a evitar confusiones relacionadas con la terminología. En particular, deben distinguirse los siguientes términos, sobre los que tampoco hay unanimidad respecto a su significado:

- **Oración o frase** (*sentence*): es un conjunto de palabras con las que se expresa un sentido gramatical en un lenguaje determinado.
- **Proposición** (*proposition*): es lo que una oración declarativa expresa, es decir, su significado, y que puede ser verdadero o falso.
- **Sentencia** (*statement*): es una proposición referida a una determinada entidad, aunque también es considerada por algunos autores como sinónimo de proposición.

La distinción entre oración y sentencia fue propuesta por Strawson (1950), quien además prefirió el uso de este último término frente al de proposición. Por otro lado, Baber (2011) propone una sutil diferencia entre proposición y sentencia: dos oraciones dan lugar a la misma proposición si dicen (significan) lo mismo, mientras que dos oraciones dan lugar a la misma sentencia si dicen lo mismo sobre la misma cosa.

Desde nuestro punto de vista, la definición de Baber se corresponde perfectamente con el concepto de sentencia al que hace referencia la Definición 6.2 por

las siguientes razones:

- Aclara las diferencias entre sentencias y oraciones. Una vez más hacemos hincapié en el hecho de que un protomodelo trata de entidades no materiales. Al igual que los conceptos, las sentencias son entidades abstractas que a su vez son usadas por el agente modelizador para relacionar otros conceptos. Es muy común confundir las expresiones con lo que significan. Nuestro enfoque va más allá puesto que afirmamos que un protomodelo puede existir sin ser expresado en un lenguaje (es decir, sin que las sentencias tengan unas oraciones asociadas).
- Una sentencia de un protomodelo, por el hecho de ser una sentencia, debe enunciarse de tal manera que su veracidad pueda ser comprobada respecto a *algo*. En el contexto de protomodelo, ese *algo* es la entidad que hemos llamado *original*. Por tanto, las sentencias de un protomodelo no pueden ser cualquier sentencia. Si fuesen expresadas en lenguaje natural deberían cumplir algunas condiciones para considerarlas como tales:
  - Tienen que ser declarativas, y por tanto quedan excluidas por ejemplo las frases interrogativas o imperativas.
  - El sujeto debe ser un pronombre personal en tercera persona que se pueda sustituir por *la entidad que va a ser evaluada como original del protomodelo* o por alguna de sus características (el concepto de característica, introducido en el ejemplo de mesa, será tratado en detalle en el siguiente apartado). Por ejemplo la frase *Juan salió del coche* no puede dar lugar a una sentencia de un protomodelo, puesto que simplemente enuncia un hecho sobre una entidad concreta (Juan) y no puede ser usada para ser evaluada sobre otras entidades. Como consecuencia, las frases impersonales tampoco pueden ser sentencias de un protomodelo.
  - No ser ambiguas. Debe estar claro qué están afirmando y a qué están referidas.
  - Deben ser objetivas, es decir, su evaluación no debe depender del sujeto que realice la validación. Por ejemplo la frase *tiene buen sabor* no puede ser objetivamente evaluada como verdadera o falsa puesto que expresa una opinión dependiente del que la evalúa.
  - Si no se especifica un determinado contexto, deben ser independientes

> del tiempo y el espacio, es decir, su veracidad o falsedad no dependerá del momento o el lugar en el que se validan.

No tener en cuenta estas reglas puede dar lugar a confusiones. Así por ejemplo en Gonzalez-Perez and Henderson-Sellers (2007) podemos encontrar la siguiente afirmación:

> We could call the sentence "today it's raining" a model, but it would be of little use.

Estamos de acuerdo en que dicha frase sería poco útil, pero por las razones indicadas, no puede dar lugar a una sentencia de un modelo, ni de un protomodelo tal como lo hemos definido, puesto que esa sentencia simplemente está declarando un hecho concreto en un momento concreto.

Veamos algunos ejemplos correctos de sentencias de protomodelo. Considérese el protomodelo de los números pares. El protomodelo estaría formado por la sentencia: d1="(la entidad para ser evaluada como original del protomodelo *número par*) es un número natural" y d2="el resto al dividirlo por dos es cero".

Expresado matemáticamente, las sentencias de la función de validación serían d1="$x \in \mathbb{N}$)" y d2="x, x mod 2 = 0)".

Otro ejemplo interesante, tomado del ámbito del software es el protomodelo de fecha en el que la función de validación podría tener más o menos sentencias en función del grado de precisión deseado. Si, por simplicidad, omitimos los detalles relativos a los años bisiestos, las sentencias serían de la forma d1="(la entidad para ser evaluada como original del protomodelo *fecha*) tiene 3 números: d, m, y"; d2="y (que hace referencia a una característica de fecha) es un entero"; d3="m es un entero entre 1 y 12"; d4="d es un entero entre 1 y 31 si m es 1, 3, 5, 7, 8, 10 or 12"; d5="d es un entero entre 1 y 30 si m es 4, 6,9 or 11" y d6="d es 28 si m es 2".

Finalmente consideremos el protomodelo de Sol. En este caso nuestra intención no es mostrar las sentencias sobre el modelo de sol para las que serían necesarios conocimientos de astrofísica, sino indicar un ejemplo en el que un protomodelo sólo tiene un original. En la sección 6.7.4 describiremos los modelos derivados de estos protomodelos y que denominaremos *modelos unitarios* (*singleton*).

6.1.4. Características de un protomodelo

Para que una sentencia de un protomodelo pueda ser evaluada como verdadera o falsa respecto a una entidad debe hacer referencia a propiedades o características que puedan ser percibidas en esa entidad, así como a relaciones entre estas características. A esas propiedades se les denomina *características del protomodelo*.

Al igual que ocurre con concepto y sentencia, ha habido muchas discusiones filosóficas sobre el verdadero significado de *característica o propiedad* de una entidad (Chierchia and Turner, 1988; Swoyer and Orilia, 2011).

En el contexto de nuestra teoría sobre modelización deben tenerse en cuenta las siguientes observaciones:

- Las características son como *contenedores* de valores usados para validar si una entidad es o no original del protomodelo. Por tanto cada entidad debería proporcionar algún valor para cada característica con el fin de que las sentencias del protomodelo puedan ser evaluadas.
- Nuevamente debe tenerse en cuenta que las características son objetos abstractos que no deben confundirse con las características físicas de un objeto. Las características son los elementos en los que se basa el modelizador para crear las sentencias y deben ser abstractas como éstas.
- La sentencia más simple es la que hace referencia únicamente a una característica, pero pueden construirse sentencias más complejas en las que se establezcan relaciones entre características, características que hagan referencias a otros protomodelos, características referentes al propio protomodelo, etc.
- Cuando una entidad es validada para comprobar si es un original de un protomodelo se hace uso de una correspondencia (mapping) entre las características del protomodelo y los valores concretos de cada característica para ese original. Si además las sentencias del protomodelo establecen relaciones entre las características, entonces esas relaciones también se deben mantener entre los valores del original. Esta propiedad, denominada *homomorfismo*, ha sido estudiada por diferentes autores en diferentes contextos, por ejemplo, (Klir, 2001) para sistemas o (Kurtev, 2007) para modelos (protomodelos según nuestra terminología).
- Potencialmente, sobre una entidad pueden diferenciarse un conjunto muy elevado de características. El proceso de abstracción consiste en distinguir

un subconjunto de esas características en diversos objetos conceptualizándolas.

Si volvemos a los ejemplos anteriores diremos por ejemplo que *pata* es una característica del modelo de mesa o que *d,m,y* son características del modelo de fecha.

6.1.5. Sistemas, temas (subjects) y originales

Una cuestión terminológica es la elección del término *original* para referirnos a las entidades validadas por un protomodelo. Este término fue utilizado por Stachowiak (1973) para referirse a las entidades modelizadas y hacía referencia a que el original es anterior al modelo en el sentido de que el modelo procede de él. Esto, tal como apunta Kühne (2006a) no es aplicable a los modelos que generan *originales* (el original procede del modelo). No obstante, hemos preferido utilizar la terminología de Stachowiak puesto que el término *original* también tiene la acepción de *algo nuevo o diferente a otros*, que sí sería aplicable al segundo caso en el sentido de que los originales de un modelo se distinguen de los originales de otros modelos.

Seidewitz, en la definición que nos ha servido como punto de partida, utiliza la expresión *system under study (SUS)* mientras que Gonzalez-Perez and Henderson-Sellers prefieren el término *subject*.

Independientemente del término o expresión elegida para la definición de modelo, la gran mayoría de los autores termina refiriéndose a la entidad modelizada como *sistema* tal como se puede comprobar en el RCM de modelo.

Desde el punto de vista de las definiciones de protomodelo y original (Definiciones 6.2 y 6.3) es importante recalcar que aún admitiendo que en la práctica la mayoría de los originales son sistemas, tales definiciones son válidas también para entidades que no son sistemas. La única condición que debe cumplir esa entidad no material es que pueda realizarse una evaluación de las sentencias del protomodelo sobre ella. Las implicaciones que se derivan de que un original también sea un sistema serán abordadas en el siguiente capítulo (apartado 7.2.3).

6.1.6. Triángulo de la modelización y protomodelos

En el capítulo anterior indicamos la importancia del triángulo de la modelización y el principio de independencia a la hora de abordar la definición de modelo. Sin

embargo, parece contradictorio que ya en la definición de protomodelo hayamos utilizado el lenguaje natural o el matemático para expresar las sentencias del protomodelo. En el capítulo 4 se indicó que diversos autores incluyen el concepto de lenguaje en la definición de modelo y esa interdependencia es, en nuestra opinión, fuente de muchas inconsistencias y ambigüedades que queremos evitar desde el principio. Como describiremos en detalle en el capítulo 8, las relaciones entre modelo y lenguaje son muy complejas pero ambos conceptos pueden ser abordados por separado teniendo en cuenta que modelo es más general que lenguaje y debe ser tratado en primer lugar, al igual que *concepto* existe antes que el lenguaje para describirlo. Por tanto, debe tenerse presente que a pesar de que para comunicar qué es un protomodelo debemos utilizar un lenguaje (el lenguaje natural en este caso), un protomodelo no necesita de un lenguaje para existir. En nuestra propuesta, los protomodelos pueden ser construidos por el modelizador de manera abstracta previamente a su expresión en algún lenguaje e incluso no ser expresados nunca.

Las condiciones que deben cumplir las sentencias descritas en la sección 6.1.3 pueden dar la impresión que están referidas a un lenguaje (el lenguaje natural), no obstante las mismas condiciones podrían ser expresadas refiriéndose a otro lenguaje (por ejemplo, el lenguaje matemático). El uso de un lenguaje para expresar las sentencias únicamente es debido a la necesidad de expresarlas para comunicarlas a un tercero. El lenguaje utilizado no es relevante siempre que permita expresar los conceptos que se quieren transmitir. Conforme a nuestro punto de vista respecto al problema de la relación entre conceptos y lenguaje indicado en el apartado 6.1.2, debe tenerse presente que el modelizador construye el protomodelo *mentalmente* incluso aunque de manera casi inmediata obtenga la expresión de esas construcciones mentales y por tanto, aunque habitualmente lo expresa mediante un lenguaje no es una condición necesaria para poder modelizar. No obstante, la influencia del lenguaje podría llegar a ser importante. En ocasiones las connotaciones asociadas a las palabras impiden comprobar si las sentencias del protomodelo son suficientes para que su función de validación sea cierta. Por ejemplo si se cambiase el nombre del modelo *mesa* por *modelo1* la evaluación de las sentencias debería producir el mismo resultado al validar cualquier entidad y no porque se sepa que se trata de una mesa debido al conocimiento previo del significado de ese término.

Como conclusión de lo expuesto en esta sección, podemos afirmar que lo que

algunos autores definen como modelo, en nuestra opinión es en realidad una aproximación a modelo: un protomodelo. A diferencia de la definición de Seidewitz, para nosotros un modelo no es simplemente un conjunto de sentencias sobre un sistema. A continuación, veremos que la modelización se apoya en la aplicación del método científico (descrito en la sección 5.2) así como el método científico también se basa en la modelización.

6.2. El proceso analítico. Modelos analíticos.

Como vimos en la sección 5.2, la observación y la experimentación son utilizadas en el método científico para el enunciado de hipótesis (análisis) y a partir de éstas realizar predicciones mediante deducción (síntesis) para posteriormente realizar un proceso de evaluación de las hipótesis. De igual forma, el proceso analítico y sintético (o al menos uno de ellos) jugará un importante papel en la construcción de las sentencias de un protomodelo. En nuestra opinión, y ésta es una de las principales contribuciones de nuestra teoría, tanto análisis como síntesis (o al menos uno de ellos) debe ser incluido en la definición de modelo, con la misma relevancia que la validación, la cual como hemos visto ha sido considerada por diversos autores y es el aspecto central de la definición de lo que nosotros hemos llamado protomodelo. De esta forma, el aspecto de utilidad del modelo cobra gran importancia en nuestra teoría: ¿puede ser útil un modelo si no puede ser usado para describir (analizar) y/o prescribir (sintetizar)? Por otra parte, la inclusión de la noción de análisis en la definición proporcionará a la noción de modelo otra característica enriquecedora: el proceso de generación real del modelo en sí mismo, que puede ser utilizado en su posterior refinamiento. Veamos cómo se produce ese proceso.

En la sección 5.2 sobre el método científico se mostraron los principales pasos que sigue un científico para obtener leyes científicas a partir de hechos demostrados por la comprobación empírica o por la experimentación. Uno de los pasos principales del método es el de análisis. En esta sección, mostraremos que uno de los métodos para la creación de un protomodelo sigue un proceso similar al del análisis en el método científico.

Desde el punto de vista de la modelización, el análisis es el proceso por el que el modelizador desarrolla un protomodelo que dará lugar a su correspondiente modelo. El proceso analítico sigue los siguientes pasos:

1. El modelizador selecciona un conjunto de entidades denominado *muestra* dentro de un conjunto de entidades similares (o relacionadas) que acabarán siendo el *dominio del modelo* y que se corresponde con el ámbito de estudio del método científico. El dominio del modelo permite discriminar las entidades sobre las que se pretende obtener un modelo. El modelizador selecciona una muestra representativa generalmente por la imposibilidad de observar el conjunto total de entidades del dominio.
2. El modelizador analiza las entidades de la muestra y mediante un proceso de abstracción distingue, bien mediante observación directa o ayudándose de instrumentos, las características que son de interés para una finalidad determinada.
3. El modelizador mediante inducción propone un conjunto de sentencias para que constituyan la función de validación del protomodelo, tal como se ha definido en la sección anterior. La clave del proceso analítico es que el modelizador no obtiene generalmente ese conjunto de sentencias en un primer intento ni las enuncia de una forma aleatoria sino que son tratadas como hipótesis, por analogía con el método científico, que deben ser contrastadas para poder ser consideradas como sentencias del protomodelo.
4. El modelizador comprueba la validez de las hipótesis para cada una de las entidades de la muestra. Es importante resaltar que se cumple lo indicado por Favre (2004a) respecto a quien *tiene la verdad* (si el modelo o la entidad modelizada) y que podríamos denominar las *reglas de la verdad de Favre*: en el caso analítico las entidades de la muestra del modelo representan *la verdad*, es decir, que si una determinada hipótesis resulta ser falsa para alguna o todas las entidades de la muestra, entonces dicha hipótesis debe ser descartada y, si es posible, reemplazada por otra hipótesis que pudiera ser cierta. Se trata, por tanto de un proceso iterativo en el que unas hipótesis no válidas se reemplazan por otras hasta obtener un conjunto de hipótesis que sean válidas en la muestra. Debe tenerse en cuenta que si el dominio del modelo tiene un número finito de entidades y es posible validar las hipótesis sobre el conjunto completo entonces el proceso finaliza una vez validadas todas las entidades y por tanto la muestra coincide con el dominio del modelo. Sin embargo, si el dominio tiene un número infinito de entidades la muestra está contenida en el dominio del modelo y entonces el proceso finalizará cuando el modelizador considere que ha analizado y validado una

muestra representativa. Estos dos casos se corresponden con la inducción completa e incompleta descrita en el método científico.

Otra cuestión importante es que el modelizador no sólo obtiene las hipótesis sino que también elige la muestra del dominio sobre la que va a trabajar para obtener el protomodelo. Esto quiere decir que no sólo deben ser ciertas todas las hipótesis para cada uno de los elementos de la muestra, sino que también deberían ser falsas para las entidades que el modelizador ha excluido del dominio del modelo que pretende obtener. En ese caso las hipótesis serían demasiado generales como para considerarlas sentencias del modelo asociado a ese dominio. Para evitarlo, el modelizador debería obtener unas hipótesis más restrictivas.

5. Tanto si se valida una muestra representativa como el dominio completo del modelo, una vez terminado el proceso iterativo, las hipótesis pasan a ser las sentencias de la función de validación del protomodelo resultante. Así mismo, en ese momento las entidades de la muestra pueden ser llamadas *originales*.

Como ya se indicó respecto al método científico, no existe *un proceso analítico* válido para todos los casos, entendido como un único conjunto de reglas que permitan a cualquier modelizador en cualquier dominio elegir las hipótesis y las muestras para obtener un protomodelo. Por ejemplo, un dentista que observa una muestra de caras podrá obtener un protomodelo de cara con muchas características relacionadas con la boca y los dientes. La misma muestra observada por un oftalmólogo daría como resultado un protomodelo de cara con características relacionadas con los ojos. En todos los casos, los protomodelos resultantes dependen de cómo percibe la muestra el modelizador y con qué finalidad.

Otro ejemplo interesante del ámbito de las TIC, es el del procesamiento de un determinado esquema relacional mediante herramientas de ingeniería inversa con el fin de obtener (proto)modelos conceptuales, constituidos bien por clases relacionadas (expresadas comúnmente en UML) o por entidades y relaciones (expresadas utilizando diagramas E/R).

Aunque el modelizador seguirá iterativamente los pasos del proceso analítico que hemos descrito de una manera genérica (observación, abstracción, inducción, validación) el modo de obtener las hipótesis será diferente en cada caso y en general, diferentes modelizadores obtendrán diferentes protomodelos. El proceso particular de obtención de las hipótesis lo llamaremos *constructor analítico.*

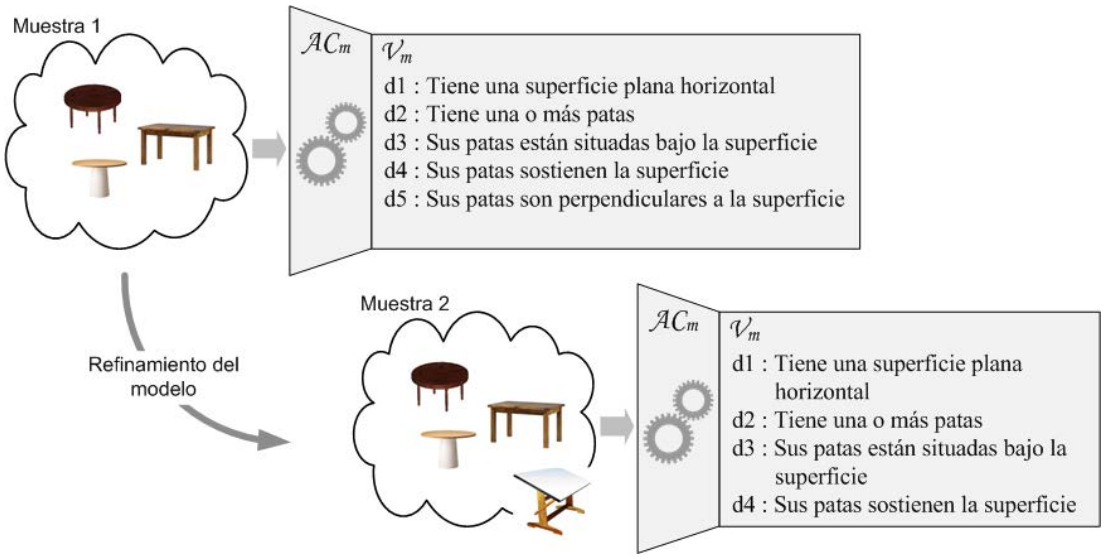


Figura 6.1: Ejemplo modelo analítico

Queremos resaltar que lo más importante de este proceso es que el modelizador elige las hipótesis sobre una muestra y al hacerlo el modelizador asocia el protomodelo a los detalles del proceso analítico. De hecho, en nuestra opinión, el proceso analítico ha permitido obtener al modelizador algo más rico que un protomodelo que ahora sí podemos llamar *modelo*, el cual además de las sentencias de su protomodelo incluye las peculiaridades del proceso analítico.

El proceso analítico es lo que permite distinguir un protomodelo de un verdadero modelo. En la figura 6.1, representamos esquemáticamente el ejemplo del modelo de mesa. En esta figura se observa como en una primera aproximación el modelizador obtiene cinco hipótesis que tendrá que validar con un conjunto de entidades del dominio (muestra 1) formado por mesas de diferentes tamaños y con diferente número de patas, y que efectivamente cumplen las hipótesis para todas ellas. En su análisis el modelizador ha identificado unas características determinadas (la superficie, las patas, su posición relativa, etc.) mientras que ha ignorado (por abstracción) otras características como el color, sus dimensiones o las propiedades de los materiales usados para su construcción.

Posteriormente utiliza otra muestra (ampliación de la primera) dentro del dominio (muestra 2) para la que una de las sentencias, la d5, no es válida y por tanto debe ser descartada al observar que una de las mesas de la nueva muestra no tiene sus patas perpendiculares a su superficie. El resultado es que la función de validación final queda con 4 sentencias. Este ejemplo ilustra la importancia de elegir una muestra que sea realmente representativa cuando no

es posible realizar la validación del dominio completo. Por otro lado, también permite ilustrar que el constructor analítico no es algo que tenga importancia únicamente al construir el protomodelo por primera vez, sino que como parte del modelo, permite su refinamiento continuo. Una vez más, no debe olvidarse que en este caso quiénes tienen la última palabra son los originales y son éstos los que determinan si un modelo es válido o no. Dicho de otro modo, el modelo mediante su constructor analítico se *adapta* a los originales a los que modeliza. Mientras que un protomodelo es estático, el modelo es algo vivo en el que las sentencias pueden ir refinándose a medida que se detectan nuevos originales para los que no eran válidas las hipótesis iniciales o incluso habiendo validado todos los originales del dominio se comprueba que las hipótesis eran demasiado generales y debe ser sustituidas por otras más específicas.

También podría ocurrir que en lugar de cambiar la muestra, cambiasen los originales en sí mismos, lo que forzaría a un reanálisis del modelo. Ése sería por ejemplo el caso del modelo del virus de la gripe. Es bien conocido que el virus de la gripe experimenta mutaciones de manera periódica. Una mutación fuerza el reanálisis del modelo del virus de la gripe de tal manera que pueda seguir siendo válido para el desarrollo efectivo de nuevas vacunas. Dicho reanálisis puede ser realizado con las mismas técnicas de análisis utilizadas para su actual constructor analítico u otras si éstas dejan de ser las adecuadas para los nuevos originales. En cualquiera de los casos, nos encontramos con un nuevo modelo en el que ha cambiado el dominio, las sentencias, e incluso el constructor analítico. Curiosamente el modelo anterior *se queda sin los originales físicos* pero permanecen los abstractos (ver la sección 6.7.1 donde se trata la naturaleza física o abstracta de los originales).

Debe observarse que una de las utilidades evidentes de la obtención de este tipo de modelos es la de *sustituibilidad*, es decir, la capacidad que tiene el modelizador de sustituir a cualquiera de los originales del modelo por éste para un determinado fin. En algunos casos además los modelos son el único modo de estudio de dichos originales por la imposibilidad de acceder directamente a ellos. Tal es el caso de los modelos de planetas o el modelo del Sol. Como hemos visto, en todos ellos el modelo se *adapta* a los originales pues éstos existen antes que él. Por tanto, el objetivo de este tipo de modelos es describir a sus originales mediante un proceso de análisis para así poder utilizar esos modelos con algún fin en lugar de usar directamente sus originales.

Llamaremos *modelos analíticos* a los modelos obtenidos mediante un proceso de análisis.

Definición **6.5.** Modelo analítico
Se define **modelo analítico** como un par $(\mathcal{AC}_m, \mathcal{P}_m)$ compuesto por (1) el proceso iterativo específico, denominado *constructor analítico* $\mathcal{AC}_m$, por el que un modelizador crea un protomodelo indicando, seleccionando y comprobando la validez de un conjunto de sentencias, denominadas *hipótesis* sobre un conjunto de entidades, denominado *muestra* y (2) el protomodelo $\mathcal{P}_m$ resultante del proceso iterativo. Una vez que todas las hipótesis han sido verificadas para todos los elementos de la muestra pasan a ser las sentencias del protomodelo o simplemente *sentencias del modelo*. Todos los términos introducidos previamente para protomodelo pueden ser aplicados al modelo resultante: sentencias del modelo, originales del modelo, características del modelo, etc.

De esta definición se observa que un modelo analítico tiene únicamente un $\mathcal{AC}_m$. Esta restricción resalta que la diferencia entre un modelo analítico y un protomodelo es el constructor analítico. El modelo analítico se construye de una forma única especificada en su $\mathcal{AC}_m$ el cual forma una parte intrínseca de él. Esto no impide que otro modelizador pueda obtener el mismo protomodelo usando un $\mathcal{AC}_m$ diferente, aunque para nosotros sería un modelo diferente puesto que fue construido (y podrá ser posteriormente refinado) de manera diferente.

6.2.1. Constructor analítico mediante ejemplo. Prototipos

El constructor analítico tiene lógicamente una enorme importancia para comprender la diferencia entre un protomodelo y un modelo analítico. Sin embargo, es habitual identificar el protomodelo con el modelo analítico sin prestar atención a su constructor. La principal razón de este hecho es que mientras el protomodelo podría ser expresado sencillamente en un lenguaje (de modelización o no), el constructor analítico suele ser difícil de expresar en un lenguaje. El proceso analítico es un proceso mental innato a los humanos que incluye, tal como hemos visto, otros procesos cognitivos como la abstracción, inducción, inferencia, identificación de características, etc. La capacidad de análisis es una habilidad con la que nacemos. Sin embargo, algunas veces no somos capaces de explicar cómo utilizamos esa habilidad para llegar a unas determinadas conclusiones.

En algunos casos el proceso es riguroso y formal, de tal manera que puede ser

transmitido a otras personas (por ejemplo mediante lenguaje natural) o incluso en un lenguaje formal que puede ser interpretado por un sistema informático. Sin embargo, habitualmente ocurre lo contrario. Consideremos, por ejemplo, el caso de un niño que vive en una tribu primitiva donde nunca han visto una maleta y queremos transmitirle el modelo de maleta. Podemos ir enunciando sus sentencias en un modo comprensible para él, por ejemplo "tiene dos tapas unidas por algo (llamado bisagra) que permite abrirla"; "las dos tapas encajan de alguna forma para que no se salgan los objetos que quedan dentro"; "tiene un objeto sujeto en un lateral para poder llevarla con la mano"; etc. El problema es que aunque consigamos enunciar todas las sentencias de una manera comprensible, describiendo las características de una maleta, probablemente no seamos capaces de que ese niño obtenga realmente el modelo de maleta, de tal manera que pueda analizar muestras de maletas (constructor analítico) y obtener por sí mismo las sentencias.

Uno de los sistemas empleados cuando se torna imposible explicar cómo realizar el proceso analítico es el que llamaremos *Constructor analítico mediante ejemplo* ($\mathcal{AC}_m$ *by example*). Consiste en que el modelizador elige un elemento representativo del dominio denominado *prototipo del modelo* y lo transmite a un destinatario para que éste pueda obtener por sí mismo el constructor analítico del modelo. Coloquialmente esta situación se produce, por ejemplo, cuando un modelizador humano quiere transmitir un modelo a un destinatario y éste no puede comprenderlo. En nuestra opinión esa *incomprensión* ocurre porque el destinatario, aunque conozca las sentencias (el protomodelo), no puede hacer nada con ellas mientras no se le explique cómo se han obtenido. Cuando el modelizador proporciona un prototipo, es decir un original representativo del modelo, entonces el destinatario puede comprender el homomorfismo existente entre las características del modelo y los correspondientes valores en el prototipo y sólo a partir de ahí puede entender por sí mismo cómo se han obtenido las sentencias del modelo.

El ejemplo anterior nos permite ilustrar este proceso. El mejor modo de que el niño comprenda cómo se han obtenido las sentencias del modelo de maleta (su protomodelo) que le hemos transmitido es mostrándole una maleta que sea representativa , un prototipo de maleta. Elegir una maleta de diseño no sería buena idea. Con el prototipo de maleta, el niño puede comprender el proceso analítico, y por tanto su constructor, y a partir de ahí podrá refinar el modelo

como ya describimos para el caso del modelo de mesa.

Debe tenerse en cuenta que el constructor analítico por prototipo se diferencia de un proceso analítico normal en que en éste último el modelizador no sabe las sentencias de antemano sino que las obtiene mediante el proceso que describimos en la sección 6.2. Por el contrario, en el caso del constructor analítico por prototipo el modelizador ya tiene las sentencias (el protomodelo) pero desconoce el proceso para obtenerlas hasta que se proporciona un protipo. El resultado final es un constructor analítico pero el proceso para su obtención es diferente.

6.3. Modelos sintéticos. El proceso sintético.

Al igual que el análisis, la síntesis juega un papel importante en la modelización. En la sección 5.2 describimos cómo la síntesis es en cierto modo el proceso inverso al análisis y en el caso del método científico juega un doble papel: 1) como uno de los métodos para obtener las hipótesis y 2) como el método que permite obtener las predicciones a partir de las hipótesis para luego realizar la validación con o sin experimentación. En esta sección analizaremos un tipo diferente de modelos que es similar al segundo caso, dejando el primero (la síntesis de hipótesis) para la sección dedicada a la metamodelización.

En la sección anterior hemos visto cómo los modelos analíticos permiten la descripción de originales existentes mediante un proceso de análisis. No obstante, hay situaciones en las que ocurre justo lo contrario: el modelo existe previamente, antes de que existan originales, y su finalidad es justamente obtener originales de él. En este caso, citando nuevamente a Favre, la verdad está en el modelo. Es el modelo el que determina la existencia de los originales y son éstos los que se adaptan a él y no a la inversa como ocurre con los modelos analíticos.

Como vimos respecto al método científico, la síntesis consiste en volver a construir e integrar las partes de un todo. En el caso de la modelización el objetivo es obtener los originales, y para ello, tal como ocurre con el proceso analítico, podemos definir un proceso, que llamaremos *proceso sintético*, generalmente iterativo, constituido por los siguientes pasos:

1. El modelizador crea como punto de partida las sentencias de un protomodelo. Estas sentencias iniciales puede haberlas creado de diferentes formas que serán explicadas posteriormente (creativamente, por metamodelización, mediante ejemplo, etc.). En algunos casos el modelizador realiza un análisis

de unas sentencias previamente existentes para deducir las que serán las sentencias del protomodelo, tal como vimos en el proceso de síntesis del método científico.

2. El modelizador usa como materia prima inicial un conjunto de ciertos elementos, que llamaremos *semillas*, y que le sirven para particularizar las características definidas en las sentencias del protomodelo. Como ocurre con las sentencias iniciales del paso anterior, las semillas no suelen ser creadas por el modelizador, sino que las obtiene generalmente mediante análisis de entidades existentes. Tal como indicamos a propósito del método científico, análisis y síntesis son procesos complementarios.
3. Mediante deducción el modelizador obtiene una versión preliminar de una entidad sintetizada. Obsérvese que mientras que en los modelos analíticos el dominio del modelo es seleccionado por el modelizador a partir de entidades existentes, en este caso son las semillas las que condicionan el tipo de originales que se van a sintetizar. A diferencia de lo que ocurría en el proceso analítico donde el modelizador obtenía hipótesis, en este caso lo que obtiene son entidades sintetizadas. En general, esas entidades sintetizadas no pasan a ser los originales del modelo al primer intento (como tampoco las hipótesis pasan a ser sentencias del modelo en el primer intento) sino que deben ser evaluadas según las sentencias del protomodelo en un proceso iterativo.
4. En caso de que la evaluación no sea satisfactoria, deberá volver al primer paso del proceso para refinar las sentencias. Dependiendo de la complejidad de las sentencias, obtener entidades sintetizadas satisfactorias puede no ser una labor trivial y por tanto deberá repetir iterativamente estos pasos hasta obtener el protomodelo deseado.
5. Si el modelizador valida satisfactoriamente la entidad sintetizada, entonces el proceso finaliza y ésta pasa a ser un original del modelo. El método obtenido para crear originales una vez finalizado el proceso sintético será un *constructor sintético* del modelo. A partir de ese momento, mediante la utilización de un constructor sintético y diferentes semillas, el modelizador podrá sintetizar diferentes originales, siendo todos ellos válidos respecto al protomodelo obtenido en el proceso.

Al igual que en los modelos analíticos, todos los originales tienen en común un conjunto de características definidas a través de las sentencias del modelo y que

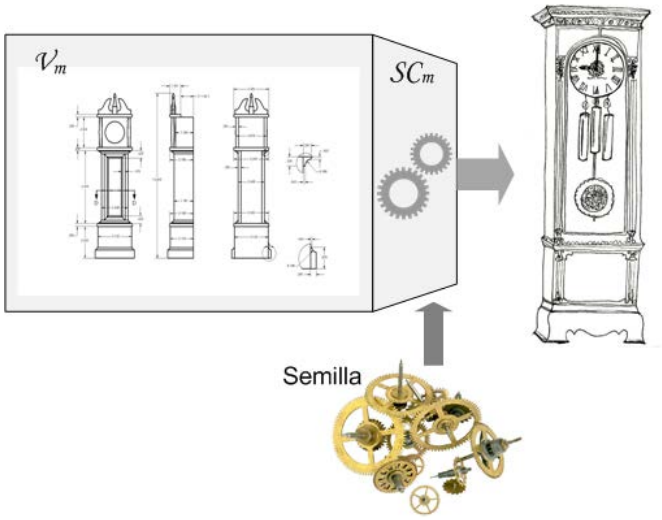


Figura 6.2: Ejemplo modelo sintético

en este caso podrían ser interpretadas como las post-condiciones del proceso de síntesis asociado con el modelo. Igualmente, las sentencias forman la función de validación que debe ser cierta para todo original del modelo.

El conjunto de entidades que forman las semillas de este tipo de modelos no es arbitrario. Cada constructor sintético necesita que sus semillas cumplan determinados requisitos para que sean adecuadas para el proceso de síntesis.

Un constructor sintético, como el analítico, podrá ser más o menos complejo o preciso. En algunos casos la formulación del proceso carece de precisión mientras que en otros casos se dispone de un algoritmo riguroso (que es la situación deseable para por ejemplo los modelos MDE).

Veamos un sencillo ejemplo. Consideremos el diseño de un nuevo modelo de reloj, representado esquemáticamente en la figura 6.2. El diseñador de relojes seguirá el proceso sintético creando unas especificaciones iniciales (sentencias del protomodelo inicial), e irá ideando los pasos necesarios para su construcción a partir de unas piezas imaginarias (las semillas) iterativamente hasta conseguir un constructor sintético que le permita obtener los originales de su modelo de reloj. Posteriormente podrá construir un original físico a partir de ese original obtenido mediante su modelo.

Una vez más debe insistirse en el hecho de que tanto modelos, originales así como las semillas son entidades abstractas, a pesar de lo que puede parecer al observar el ejemplo de la figura. Obsérvese que en este caso se ha recurrido a una representación gráfica de las sentencias del modelo en forma de planos de

construcción, mientras que las semillas se han representado como piezas a partir de las cuales se construye el reloj, únicamente a efectos de ilustrar el ejemplo para una mejor comprensión del proceso. Es muy común confundir en general un modelo con su expresión mediante un lenguaje (en este caso gráfico), pero en el caso de los modelos obtenidos mediante un proceso sintético también podríamos confundir el modelo con su construcción física. En la sección 6.7 profundizaremos sobre las implicaciones entre lo abstracto y lo físico en la modelización.

Al igual que con los modelos analíticos, queremos resaltar que un constructor sintético forma parte intrínseca del modelo puesto que el modelizador elige las semillas y las sentencias y al hacerlo el modelizador asocia el protomodelo a los detalles del proceso sintético. El modelo es igualmente mucho más que un protomodelo, pues incluye el proceso de creación de originales a partir de las semillas y las sentencias. Este proceso sintético, tal como hemos visto, permite el refinamiento continuo del modelo, pero en este caso, en virtud de las *reglas de la verdad* de Favre (2004a), quienes tienen la última palabra son las sentencias y sus semillas asociadas. Ahora son los originales, sintetizados mediante el constructor sintético obtenido, los que se *adaptan* a las sentencias y las semillas. Esta distinción es clave para comprender las diferencias entre los procesos analítico y sintético.

Llamaremos *modelos sintéticos* a los modelos obtenidos mediante un proceso sintético.

Definición **6.6.** Modelo sintético
Se define **modelo sintético** como un par $< \{\mathcal{SC}_m\}, \mathcal{P}_m >$ compuesto por (1) un conjunto no vacío de *constructores sintéticos* que tienen como finalidad la creación de nuevos originales del modelo a partir de conjuntos de otras entidades denominadas *semillas del modelo* y (2) el protomodelo $\mathcal{P}_m$ compuesto por las sentencias que constituyen las poscondiciones de $\mathcal{SC}_m$.

En la Definición se observa que, a diferencia de los modelos analíticos, un modelo sintético puede tener más de un $\mathcal{SC}_m$. Efectivamente, mientras que el modelo analítico se construye de una forma única a través de su constructor analítico, los modelos sintéticos pueden especificar diferentes formas de construir sus originales siempre que todas ellas satisfagan las sentencias de su protomodelo.

6.3.1. Constructor sintético mediante prototipos

La síntesis, como el análisis, también es un proceso mental innato a los humanos que incluye, tal como hemos visto, otros procesos cognitivos como la abstracción, deducción, inferencia, razonamiento lógico, etc. Igualmente puede ser un proceso difícil de describir en un lenguaje o por el contrario puede ser formal y preciso. Ejemplos de constructores sintéticos expresados en lenguajes más o menos formales son más fáciles de conseguir que en el caso de los modelos analíticos. Por ejemplo, la obtención de la fila de una tabla a partir del modelo de datos conceptual y un conjunto de semillas (los valores concretos para cada columna) en una base de datos relacional. En otros casos no es tan sencillo expresar un constructor sintético, al menos inicialmente. Ése sería el caso del cliente que encarga a un desarrollador informático la creación del modelo de un producto de cierta complejidad cuya fabricación estará dirigida por un sistema informático. En ese caso, el cliente puede tener dificultades para explicar al desarrollador de una manera precisa cómo será ese modelo, especialmente en lo que respecta al proceso de fabricación.

En un modelo sintético las sentencias existen antes que los originales. Cabe preguntarse *de dónde salen esas sentencias*. Como hemos indicado al describir el proceso sintético que da lugar a un constructor sintético de un modelo, esas sentencias pueden ser producto de una invención *desde cero* pero también producto de otros procesos que analizaremos posteriormente, tales como la analogía (sección 6.5) o la metamodelización (sección 7.2).

Uno de esos métodos, que permite explicar de forma intutitiva cómo realizar el proceso sintético y que tiene mucho en común con el descrito en la sección 6.2.1, es el que denominaremos *Constructor sintético por prototipo* ($\mathcal{SC}_m$ by example). Consiste en que el modelizador *imagina* una entidad que denominaremos igualmente *prototipo*, y a partir de ahí, deduce las características y las sentencias necesarias para sintetizarla mediante un homomorfismo, que es el inverso al descrito para el caso de los modelos analíticos mediante ejemplo. Obsérvese que el modelizador obtiene este constructor sintético por el proceso sintético iterativo que hemos explicado y realiza un análisis del prototipo similar al análisis de muestras que se realiza en un proceso analítico, solo que en este caso la muestra es un original ficticio que sirve de germen para obtener las sentencias en lugar de una muestra de originales existentes. Como hemos visto, el proceso iterativo consiste en la adaptación de las sentencias para permitir obtener dicho prototipo. Una vez

obtenidas las sentencias del modelo, que lógicamente son satisfactorias para el prototipo pues han sido obtenidas a partir de él, el constructor sintético permite, mediante el proceso inverso, obtener el prototipo. El modelizador realiza esta prueba de fuego para asegurarse que el método que ha ideado (este constructor sintético junto con un determinado conjunto de semillas) le permite obtener el prototipo que había imaginado originalmente. El proceso finaliza con la elección de diferentes semillas para comprobar la síntesis de otros originales y cómo éstos también cumplen la función de validación obtenida.

Como ejemplo podemos retomar el caso del programador que ha de crear un sistema informático para gestionar la fabricación de un nuevo producto. Para resolver el problema, el cliente proporciona al programador un ejemplo sobre cómo sería el producto que quiere obtener imaginando un caso particular. Éste sería el prototipo que el programador utilizará para a partir de ahí, junto con los requisitos que le ha transmitido el cliente (las sentencias del protomodelo) obtener un constructor sintético y por tanto el modelo. El programador sabrá que ha concluido su trabajo cuando sea capaz de obtener de nuevo el prototipo sintetizándolo a partir del constructor sintético que ha obtenido y el cliente validará el modelo probando adicionalmente diferentes semillas. Si alguno de los originales obtenidos no satisface las necesidades del cliente entonces el programador (que actúa como modelizador) tendrá que comenzar un proceso sintético, eventualmente utilizando otro prototipo, hasta obtener el resultado deseado.

Este método genérico de realización de un constructor sintético mediante ejemplo ilustra un hecho relevante, que es la estrecha relación que existe entre las sentencias del protomodelo y sus constructores analítico y sintético e incluso, en algunos casos como el que acabamos de describir, entre éstos entre sí.

A continuación trataremos un último tipo de modelos que constan de ambos constructores y en los que dicha relación se pondrá aún más de manifiesto.

6.4. Modelos completos

Hasta el momento hemos visto modelos que son o analíticos o sintéticos, sin embargo es habitual que un modelizador cuando construye un modelo analítico lo enriquezca con un constructor sintético. Consideremos el ejemplo del modelo de mesa. Podríamos elegir un conjunto de semillas del tipo *tabla de 1 metro cuadrado* y *4 palos de 80 cm*. Un constructor sintético generaría un original de mesa

situando la tabla sobre los palos de tal manera que se cumpliesen las sentencias del modelo. Debe entenderse este ejemplo como una ilustración del proceso sin olvidar que tanto modelo como originales son entidades abstractas aunque posteriormente puedan relacionarse con entidades físicas como veremos en la sección 6.7. Se entiende, sin embargo que los constructores sintéticos no son máquinas físicas sino procesos mentales que permiten a los modelizadores imaginar mesas y cómo construirlas.

La situación inversa, que un modelo sintético sea enriquecido con un constructor analítico ya no es tan común, aunque sería posible si tras construir el modelo sintético se observan originales, *ya existentes*, que cumplen la función de validación y que pueden formar parte del dominio del modelo. Siguiendo con el ejemplo del modelo de mesa, tal sería el caso de un constructor de mesas (suponiendo que las ha diseñado como inventor de éstas) que posteriormente descubre la existencia de otras mesas (creadas en paralelo por otro inventor) y que analiza (mediante un constructor analítico) y compara con su protomodelo asociado para concluir que forman parte también del dominio de su modelo. No olvidar que el constructor analítico de uno y otro inventores de mesas hace que los modelos de mesa de ambos sean diferentes. Para ambos inventores las mesas no existían previamente a la construcción de sus respectivos modelos.

De manera similar podríamos especificar un constructor sintético que generase originales del ejemplo de modelo de fecha descrito anteriormente, bien especificando una terna de valores enteros que cumplan las sentencias de ese modelo, o mediante un único entero (positivo o negativo) representando el número de milisegundos desde/hasta un instante de tiempo de referencia (por ejemplo 1 de enero de 1970, 00:00:00 GMT).

Podría objetarse que el modelo de mesa o el de fecha nunca fueron analíticos pues las mesas y fechas existentes son objetos artificiales que fueron en algún momento construidos por alguien. Sin embargo, no debe olvidarse que el modelo está estrechamente vinculado al modelizador. Los modelos de mesa del primer constructor de mesas y del modelizador que analiza una mesa existente son en realidad modelos muy diferentes incluso aunque los dominios de ambos modelos sean los mismos.

Obsérvese como en estos ejemplos los modelos dejarían de ser analíticos para ser simultáneamente analíticos y sintéticos. Sus originales existían antes de la creación del modelo pero ahora pueden construirse nuevos originales con sus

respectivos constructores sintéticos.

Llamaremos a este tipo de modelos *modelos completos* (*full models*).

DEFINICIÓN **6.7.** Modelo completo
Se define **modelo completo** como una tupla $(\mathcal{AC}_m, \{\mathcal{SC}_m\}, \mathcal{P}_m)$ compuesta por un *constructor analítico* $\mathcal{AC}_m$, un conjunto no vacío de *constructores sintéticos* $\{\mathcal{SC}_m\}$ y el protomodelo $\mathcal{P}_m$ relacionado.

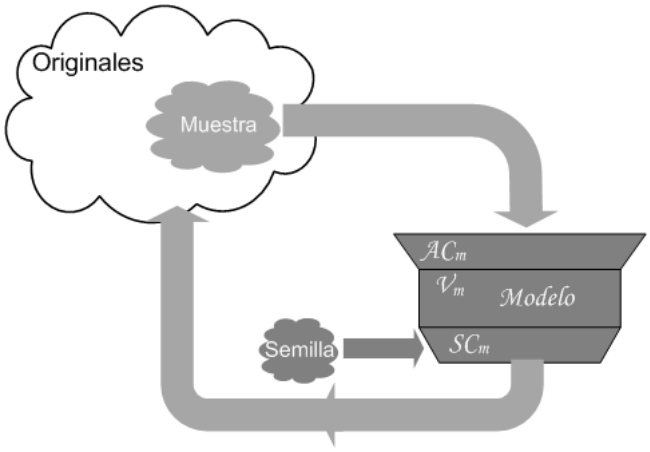


Figura 6.3: Representación esquematica de un modelo completo

La figura 6.3 representa esquemáticamente a un modelo completo. Obsérvese cómo el dominio del modelo (representado como una nube blanca) incluye tanto la muestra de originales que alimentan el constructor analítico (representados como una nube gris) como los originales creados a partir de un constructor sintético (que son el resto). De hecho, el dominio del modelo está particionado en esos dos subconjuntos de originales: los que pueden ser muestra del $\mathcal{AC}_m$ y los resultantes de cada $\mathcal{SC}_m$.

Es interesante observar cómo se siguen cumpliendo las *reglas de la verdad* de Favre (2004a) también para este caso. Por un lado los originales analizados condicionan la evolución de las sentencias del modelo de tal manera que, si éstos cambian, el modelo deberá adaptarse a ellos para que sigan cumpliendo la función de validación. Pero, por otro lado, esta evolución del modelo debe afectar a los $\mathcal{SC}_m$ que podrían sintetizar originales diferentes a los existentes. El resultado final, es que nos encontraríamos con un nuevo modelo evolución de otro anterior, del que eventualmente incluso podrían no existir sus correspondientes originales físicos (recuérdese el ejemplo del modelo de gripe) y en el que tanto los originales que son analizados por el constructor analítico como los que se han creado mediante

un constructor sintético evolucionan a la par en virtud de que tanto $\mathcal{AC}_m$ como $\mathcal{SC}_m$ deben compartir un mismo protomodelo.

En definitiva, se han mostrado tres modos de construir modelos: a partir de una muestra de entidades que es analizada (modelos sintéticos), mediante un proceso sintético que especifique como crear nuevos originales (modelos sintéticos) y mediante un modelo que combina los dos métodos anteriores (modelos completos). A continuación veremos cómo otro de los procesos, descrito en el método científico, cuya importancia es comparable a los de análisis y síntesis, ayuda a entender otro tipo de modelos.

6.5. Modelos y analogía

Hasta ahora hemos visto cómo el constructor analítico permite a los modelizadores crear un modelo. Sin embargo, hay otra forma habitual de modelizar: por analogía. El proceso es similar al proceso analítico descrito anteriormente pero con una diferencia importante. Ahora el punto de partida no es una muestra de entidades del dominio del modelo que pasan a ser originales tras el proceso de análisis, sino que se parte de un modelo diferente cuyos originales tienen características análogas a los de los originales del modelo que se quiere obtener.

6.5.1. El concepto de analogía. Modelos analógicos

La analogía ha sido estudiada por diversos autores en diversos campos como la filosofía, la semiótica, la psicología o la sociología. Se trata de un proceso cognitivo innato en el ser humano que incluso ha sido calificado como *la base del conocimiento* (Hofstadter, 2001). Mediante este proceso el ser humano relaciona dos entidades entre sí buscando semejanzas entre ellas. Sin embargo, no todas las entidades semejantes son análogas. La analogía no se basa únicamente en las semejanzas entre las características de dos entidades diferentes sino en las relaciones entre ellas. Por esa razón, la analogía está muy relacionada con los homomorfismos: se realizan correspondencias entre las características de una y otra entidad y luego se observa si las relaciones existentes entre las características de la primera entidad también se producen entre las características correspondientes de la segunda.

La analogía también está relacionada con el lenguaje y más concretamente con la semiótica (Sowa and Majumdar, 2003). La correspondencia entre concep-

tos y sus relaciones también se traslada al lenguaje y por esa razón, entidades aparentemente muy distintas tienen nombres similares o idénticos. Por ejemplo, los significados de la palabra *raíz* como parte de una planta y *raíz* como causa u origen de algo tienen su origen en el pensamiento analógico.

Consideremos por ejemplo el modelo de avión. A principios del siglo XIX un grupo de emprendedores ansiaban inventar una máquina que pudiese volar. Prácticamente la totalidad de los diferentes modelos que idearon para tal fin y que finalmente condujeron a la invención del aeroplano por los hermanos Wright, tomaron como referencia el modelo de pájaro. En efecto, no sólo existe una correspondencia entre alas de un pájaro - alas de avión, cuerpo del pájaro - cuerpo de avión, patas de pájaro - tren de aterrizaje de avión, etc., sino que también existen las mismas relaciones entre las características de uno y otro: las alas están sobre el cuerpo extendidas horizontalmente, las patas están bajo el cuerpo y permiten el aterrizaje, etc. Como puede observarse en este ejemplo, la analogía no implica una correspondencia completa (llamada isomorfismo). Por ejemplo, la propulsión de un pájaro está basada en el aleteo de sus alas mientras que en un avión las alas permanecen estáticas y es propulsado por hélices o turbinas, que no tienen correspondencia en un pájaro.

Hesse (2001) realizó una clasificación de analogía que permite entender esa falta de correspondencia completa desde otro punto de vista:

- **Analogía positiva**: selecciona aquellas características de la entidad análoga que son idénticas o muy similares a la entidad de referencia.
- **Analogía negativa**: selecciona características diferentes o poco parecidas.
- **Analogía neutral**: selecciona aquellas características para las que no existe evidencia de similitud o diferencia.

Según esta clasificación, puede haber una mayor analogía en la medida en que la analogía neutral se convierte en analogía positiva y menor si aquélla se convierte en negativa. En el ejemplo del modelo de avión, las alas, el cuerpo, etc. constituyen una analogía positiva mientras que la propulsión es una analogía negativa. En el contexto de este trabajo consideraremos que nos referimos a analogía positiva mientras no se indique lo contrario.

En el ámbito de nuestro estudio de la teoría de la modelización, la analogía también juega un papel importante puesto que permite definir un nuevo tipo de modelos cuyo constructor es diferente a los presentados hasta el momento.

Se trata de un tipo de modelo que por analogía describe entidades y eventualmente sintetiza originales que son análogos a los originales del modelo análogo que toma como *referencia*.

El modelo obtenido de este modo no es un modelo analítico aunque también sea producido mediante un proceso iterativo similar al del proceso analítico. Hay una importante diferencia respecto al modelo analítico: la muestra no existe o no puede observarse cuando comienza la construcción del modelo, es decir, no se construye el modelo a partir de la observación de una muestra de entidades del dominio del modelo. Por tanto, *la verdad* de Favre hay que buscarla en los originales del dominio del modelo de referencia.

Llamaremos *modelos analógicos* a los modelos obtenidos mediante analogía.

Definición **6.8.** Modelo analógico
Se define **modelo analógico** como un par $(\mathcal{A}g\mathcal{C}_m, \mathcal{P}_m)$ compuesto por (1) el proceso iterativo específico, denominado *constructor analógico* $\mathcal{A}g\mathcal{C}_m$, por el que un modelizador selecciona y comprueba la validez de un conjunto de sentencias, denominadas *hipótesis* enunciadas mediante analogía con un modelo denominado *modelo de referencia* y (2) el protomodelo $\mathcal{P}_m$ resultante de dicho proceso.

6.5.2. Tipos de modelos analógicos

Distinguimos tres tipos de modelos analógicos:

1. **Sintéticos**: El modelo analógico es construido mediante $\mathcal{A}g\mathcal{C}_m$ a partir de un modelo analítico de referencia, previamente existente o construido a propósito desde una *muestra de referencia* de las entidades del dominio de ese modelo de referencia, que llamaremos *originales de referencia*. En este caso, el modelo también es sintético y por tanto podrá generar sus originales a partir de cada $\mathcal{SC}_m$ y un conjunto de semillas. Como puede verse en la figura 6.4(a), esa muestra de originales de referencia no está incluida en el dominio del modelo analógico (es una muestra de otro modelo). Por otro lado los originales obtenidos por un constructor sintético del modelo analógico son originales análogos a los originales de referencia. Un ejemplo de este tipo es el modelo de avión que hemos descrito anteriormente, donde el modelo de referencia es el modelo de pájaro obtenido a partir de una muestra de pájaros y el modelo analógico es el de avión que obtendrá aviones mediante uno de sus constructores sintéticos. Los aviones son originales

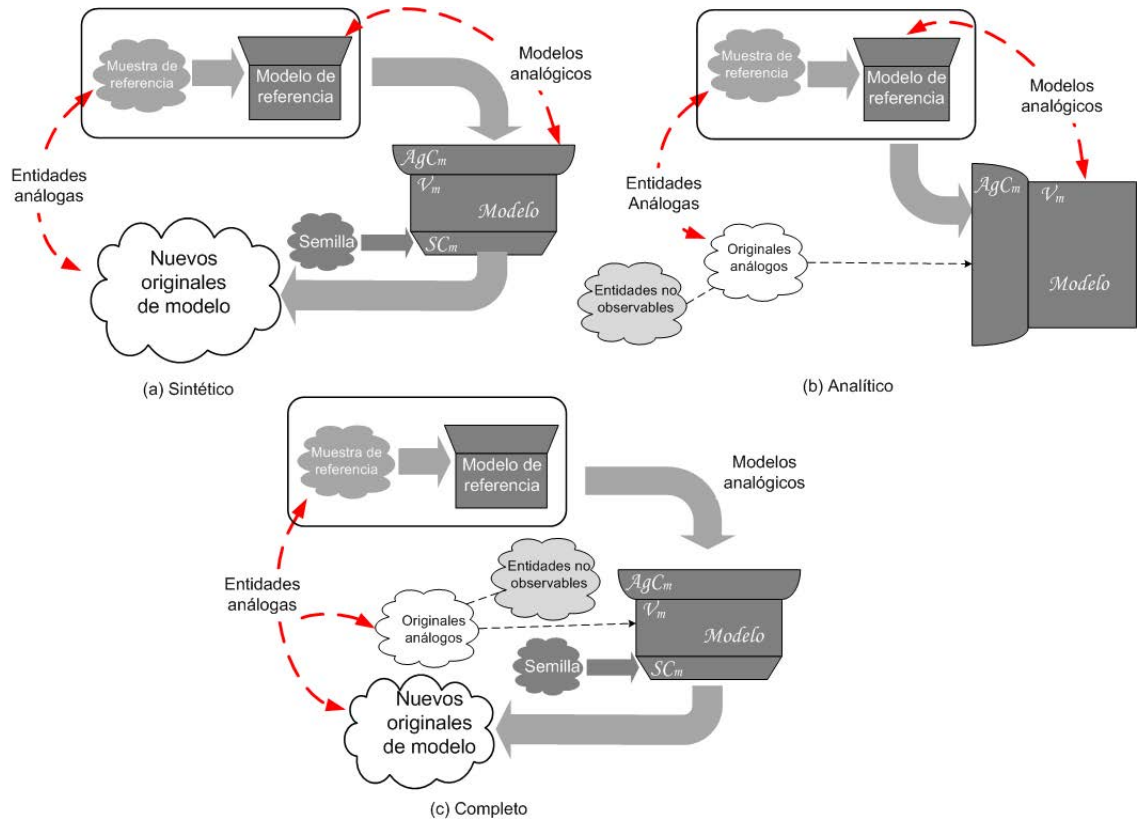


Figura 6.4: Tipos de modelos analógicos

análogos a los pájaros que son sus originales de referencia.

2. **Analíticos**: el modelo analógico es construido igualmente a partir de un modelo de referencia, pero no existe constructor sintético. Las entidades existentes no pueden ser observadas y por tanto tampoco ser utilizadas como muestra para obtener las hipótesis y posteriormente las sentencias del modelo. Sin embargo, el modelizador piensa que esas entidades no observables son análogas a los originales del modelo de referencia. Por tanto, como puede verse en la figura 6.4(b), el modelizador tiene que recurrir al modelo de referencia para construir mediante $\mathcal{AgC}_m$ las hipótesis en un proceso iterativo similar al proceso analógico. Un ejemplo de este caso es el de los modelos del átomo y del sistema planetario entre los que existe una conocida analogía (véase por ejemplo (French, 2002; Goldstone and Son, 2004)). Las hipótesis del modelo de átomo pueden formularse a partir del modelo del sistema planetario (que es su modelo de referencia) y por tanto para obtener la función de validación que pueda usarse como comprobación indirecta para validar las hipótesis sobre la muestra no observable. El

modelizador puede razonar y deducir las sentencias del modelo de átomo formulando hipótesis a partir del comportamiento del modelo planetario, dada su imposibilidad de observar los átomos. Es importante recalcar que el vínculo entre el modelo y las entidades no observables (los átomos) no es creado a partir de un constructor analítico, sino indirectamente a través del constructor analógico que toma como entrada el modelo de referencia. No obstante, cuando se produce este caso, los modelos analógicos se utilizan del mismo modo que los analíticos, es decir, son modelos descriptivos donde los originales existen antes que el modelo.

3. **Completos**: el último tipo representado en la figura 6.4(c) es una combinación de los dos anteriores: se parte de unas entidades no observables existentes que son análogas a los originales de un modelo de referencia y mediante el $\mathcal{A}g\mathcal{C}_m$ se generan las sentencias del modelo. Luego mediante cada $\mathcal{SC}_m$ se generan nuevos originales, que no olvidemos son también entidades abstractas, pero que al haber sido creadas por un constructor sintético de un modelo análogo al de referencia, estos originales son también análogos a los originales de referencia. El dominio del modelo analógico lo forman los originales obtenidos mediante un constructor sintético del modelo analógico y unos originales análogos que el modelizador induce por analogía pero que no son producto de la percepción de las entidades del dominio del modelo analógico puesto que no son observables. Es interesante indicar que los originales generados por un constructor sintético del modelo analógico permiten al modelizador *imaginar* cómo hubiesen sido las entidades no observables si hubiera sido posible percibirlas.

6.5.3. Analogía entre protomodelos y constructores. Interacciones.

Hasta el momento hemos descrito las correspondencias que existen entre las características y relaciones del modelo de referencia y del modelo analógico. Por otra parte, hemos considerado que el modelo de referencia es típicamente analítico, pero también podría ser sintético o completo, aunque como puede observarse su tipo no es relevante para que se produzca la analogía entre ambos. Desde el punto de vista del modelizador lo que se produce es una relación de *inspiración* entre ambos modelos: parece claro que de algún modo el modelizador se *inspira*

en el modelo de referencia para obtener el modelo analógico. Consideremos nuevamente el modelo de avión. Si el modelizador va a construir las sentencias del modelo de avión a partir de las sentencias del modelo de pájaro, parece lógico que no sólo encuentre sentencias del modelo de avión que son semejantes a las de pájaro sino que incluso si existen relaciones entre las sentencias de pájaro, también existan de forma parcial o completa entre las sentencias de avión. En ese caso, por la propia definición de analogía explicada anteriormente, nos encontraríamos con que habría analogía entre las sentencias de los modelos de pájaro y avión. Podríamos decir entonces que sus protomodelos también son análogos.

Lo mismo podría decirse sobre el proceso analítico y/o sintético del modelo de referencia respecto a los procesos analógico y sintético del modelo analógico. En efecto, por ejemplo si el modelo de pájaro fuese sintético la forma de construir originales y las semillas utilizadas (colocando las alas encima del cuerpo y las patas debajo) serían análogos a la forma de construir aviones y semillas de los aviones (alas colocadas encima del cuerpo y tren de aterrizaje debajo).

Por tanto, aunque no siempre tiene que ser así, el modelo analógico toma prestados algunos aspectos de las sentencias del modelo de referencia, sus constructores y sus semillas.

Podemos decir que en general existen interacciones (análogas o no) entre cada uno de los elementos (protomodelos y constructores) de ambos modelos. Una interacción interesante ocurre cuando el modelizador del modelo analógico observa primero unas entidades de referencia de las que no existe modelo. En ese caso, al no existir modelo de referencia, el modelizador lo construye normalmente pero al mismo tiempo que observa una muestra de referencia para construirlo también va identificando el homomorfismo entre los originales de referencia y los originales análogos y construye a partir de ahí el modelo analógico, no en dos pasos como hemos sugerido (primero el modelo de referencia y luego el modelo analógico análogo al de referencia), sino al mismo tiempo (primero los originales de referencia y luego los modelos de referencia y analógico simultáneamente).

6.6. Modelos

6.6.1. Definición general de modelo

Una vez definidos los diferentes tipos de modelos, podemos expresar una definición general de modelo basándonos en las definiciones anteriores.

Definición **6.9.** Modelo
Se define **modelo** como un par $(\mathcal{C}_m, \mathcal{P}_m)$ compuesto por

1. Un conjunto de constructores
$$\mathcal{C}_m = \begin{cases} \mathcal{AC}_m & \text{si el modelo es analítico} \\ \{\mathcal{SC}_m\} & \text{si el modelo es sintético} \\ \mathcal{AC}_m, \{\mathcal{SC}_m\} & \text{si el modelo es completo} \\ \mathcal{AgC}_m & \text{si el modelo es analógico analítico} \\ \mathcal{AgC}_m, \{\mathcal{SC}_m\} & \text{si el modelo es analógico sintético o completo} \end{cases}$$
2. Un protomodelo $\mathcal{P}_m$ asociado

Siendo respectivamente $\mathcal{AC}_m$, $\mathcal{SC}_m$ y $\mathcal{AgC}_m$ los constructores analítico, sintético y analógico descritos en las secciones 6.2, 6.3 y 6.5

6.6.2. Relación de modelización

En ocasiones es útil expresar que una entidad es original de un modelo o viceversa mediante una relación con el fin de estudiar sus propiedades.

Definición **6.10.** Relación de modelización
Sean dos entidades, m y o, diremos que o es un original de m o bien que m es un modelo de o, y lo denotaremos como $m \models o \iff o \in \mathcal{O}_m$

6.6.3. Universo de modelos y originales

Definición **6.11.** Universo de modelos
Se define $\mathcal{M}$ como el conjunto de todos los modelos.

Definición **6.12.** Universo de originales
Se define $\mathcal{O}$ como el conjunto de todos los originales, por lo que $\forall m \in \mathcal{M}, \mathcal{O}_m \subseteq \mathcal{O}$

La figura 6.5 están representados los universos de modelos y originales como subconjuntos de $\mathcal{E}$.

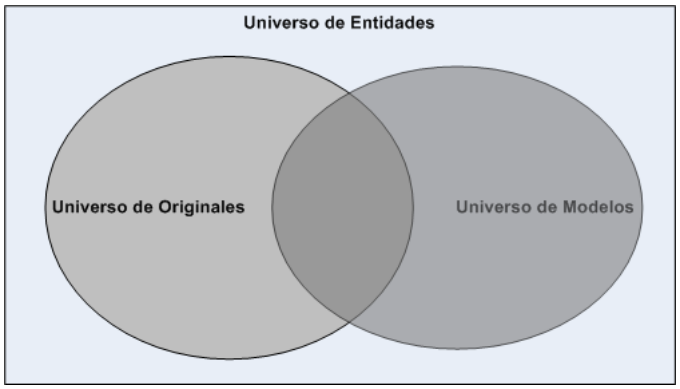


Figura 6.5: Representación de los universos de originales y modelos

6.7. Sobre la naturaleza de lo físico y lo abstracto en la modelización

Una vez descritos los diferentes tipos de modelos existentes es el momento de aclarar el problema de lo abstracto y lo físico al que hemos hecho referencia en distintas ocasiones. La confusión entre lo abstracto y lo físico en la modelización se produce con frecuencia en la literatura. Sin embargo, no se encuentran habitualmente estudios sobre este tema. Habitualmente los investigadores hacen referencia a modelos u originales físicos como si fuesen abstractos o viceversa, sin precisar que no se trata de las mismas entidades. Para nuestra teoría, por el contrario, esta distinción sí tiene una especial importancia.

Hasta el momento nos hemos referido a muestras de originales, semillas, modelos, etc., puntualizando como lo hace Klir (2001), que estos términos pertenecen a un plano conceptual y por lo tanto son abstractos (en el sentido de no materiales), sin perder de vista que en muchos casos estos conceptos tienen como referentes entidades físicas (en el sentido de materiales o reales).

La cuestión que vamos a abordar en esta sección es: ¿cuál es la relación existente entre el concepto (original, modelo, semilla) y la entidad física a la que se refiere?

Como vamos a ver a continuación nuestra teoría permite responder de una forma consistente a esta pregunta gracias al enfoque científico de la modelización.

6.7.1. Originales físicos, percibidos/concebidos/análogos y abstractos

Para responder a la pregunta planteada, revisemos nuevamente el proceso analítico para el caso de que las entidades del dominio sean entidades físicas observables. No obstante, debe puntualizarse que no tiene que ser siempre así, puesto que un modelizador puede realizar un análisis de entidades abstractas (no materiales), como por ejemplo un análisis matemático, para obtener un modelo analítico. Esas entidades abstractas, pueden ser observables pero no mediante sensores. En el caso de un modelizador humano se trataría de una observación mental. Por otro lado, hemos visto en la sección 6.5 que también pueden existir entidades físicas que no son observables.

Nos centraremos en el supuesto de que las entidades del dominio de un modelo analítico sean entidades físicas observables por ser el caso en el que intervienen más tipos de originales y por tanto se produce una mayor confusión.

Cuando el modelizador observa una entidad física obtiene en un primer momento una *imagen* de ésta a partir de sus *sensores*. En un modelizador humano esa imagen es una imagen mental y los sensores son los sentidos ayudados o no por instrumentos de medida u observación, como por ejemplo un microscopio o un micrófono de alta sensibilidad. Si el modelizador no fuese humano, obtendría igualmente una *imagen*, que en un sistema informático es típicamente digital. En ambos casos la imagen obtenida depende de las capacidades de los sensores. Así por ejemplo un modelizador humano que observa una flor a través de sus ojos obtiene una primera imagen (visual) *en bruto* de ésta, mientras que la misma flor observada por una persona ciega a través de su olfato obtendría igualmente una imagen que podríamos calificar de olfativa. En este punto, el modelizador hará uso de sus sensores en función de la finalidad que persiga para el modelo que quiere obtener. Así decidirá si obtener imágenes visuales, olfativas, auditivas, etc., o una mezcla variable de éstas y con el nivel de detalle que necesite, eventualmente ayudado por instrumentos que amplifiquen sus capacidades sensoriales naturales. Llamaremos *entidad percibida* a esa imagen, en contraste con la entidad observada que llamaremos *entidad física observable*. Mantendremos el calificativo *observable* para distinguirlas de las entidades no observables, las cuales como vimos en la sección 6.5 no pueden por su propia definición dar lugar a entidades percibidas.

Nótese que la entidad percibida no es un modelo, ni siquiera un protomodelo, es una imagen mental en bruto de la información proporcionada por los sensores

de los que dispone el modelizador.

La modelización comienza cuando el modelizador a partir de ese conjunto de datos sin sentido que forman la entidad percibida, elige y conceptualiza a través de un proceso de simplificación y abstracción aquellas características que considera relevantes para su modelo y que darán lugar a una *entidad abstracta*. Como ya explicamos en la sección 6.2 el proceso analítico continúa con el enunciado de ciertas hipótesis sobre uno o más originales percibidos que conforman la muestra elegida. Si el proceso tiene éxito las hipótesis pasan a ser las sentencias del modelo, las entidades percibidas serán los *originales percibidos* del modelo, las entidades abstractas *originales abstractos* del modelo y las entidades físicas observables pasarán a ser *originales físicos observables*.

Esta distinción entre originales es muy importante para comprender el origen de muchas de las confusiones encontradas en la literatura respecto a lo físico y lo abstracto. Para nosotros y otros autores (véase por ejemplo (Klir, 2001)), el proceso de abstracción comienza en el original percibido puesto que el modelizador no puede trabajar mentalmente con el original físico observable. Desde nuestro punto de vista, y esto es algo que consideramos un error habitual, no consideramos que la transición de un original físico observable a un original percibido sea una abstracción puesto que la pérdida de características que ocurre en ese paso sólo es debida a la incapacidad de los sensores de capturar la totalidad de características del original físico observable. Idealmente podría existir un modelizador que obtuviese *todas* las características de una entidad física observable pero eso es algo que los filósofos han debatido desde hace tiempo sin ponerse de acuerdo y que queda fuera de nuestro estudio. Obsérvese que la distinción entre físico observable, percibido y abstracto es similar a la bien conocida distinción entre dato, información y conocimiento y por tanto nuestro enfoque es consistente con los estudios sobre epistemología o teoría del conocimiento.

La figura 6.6 ilustra un ejemplo del proceso descrito anteriormente. En efecto, las entidades físicas (que pasarán a ser originales físicos observables) son los libros físicos. El modelizador percibe a través de sus sentidos diferentes libros físicos y obtiene imágenes mentales de los mismos (originales percibidos). A partir de ellos diferencia dos de sus características para obtener el modelo adecuado a una determinada finalidad: nombre del libro y autor. Las representaciones mentales de los valores concretos de esas características para los originales percibidos de la muestra en el caso de este ejemplo son: (Hamlet, W. Shakespeare) y (El Quijote,

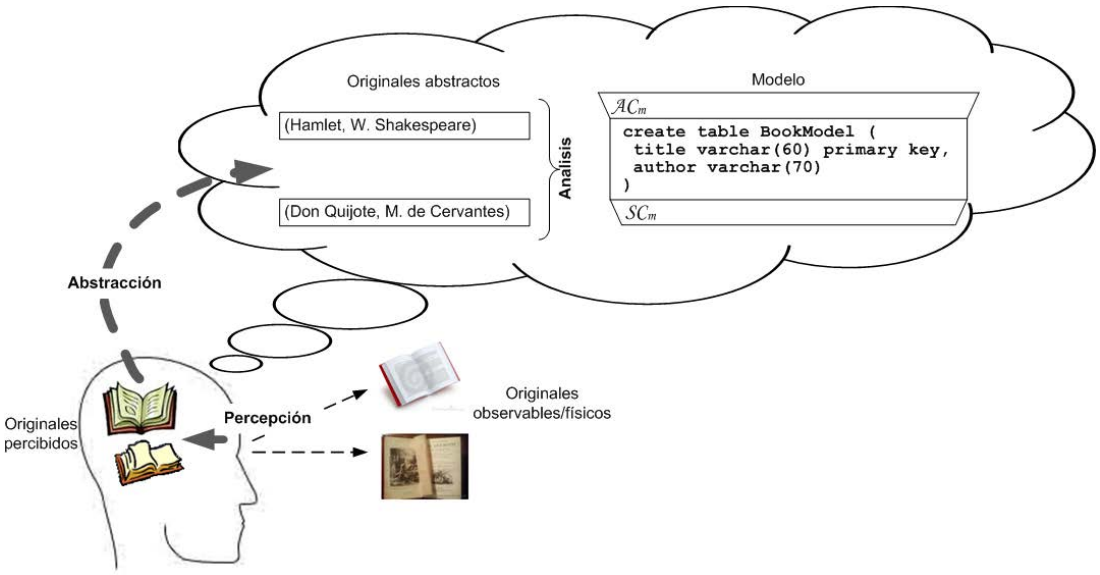


Figura 6.6: Ejemplo análisis físico y abstracto

M. de Cervantes), éstos serían los originales abstractos. El modelizador obtiene las sentencias del modelo que en el ejemplo vienen expresadas como el esquema de una tabla relacional. El constructor analítico es el proceso (mental en este caso) mediante el cual se obtienen las sentencias del modelo a partir de los originales percibidos pasando por los originales abstractos. Hemos ilustrado los diferentes tipos de originales revisando el constructor analítico, pero el proceso es similar para el caso del constructor sintético. En este caso el punto de partida para la obtención de un original percibido es un modelo sintético (o completo) y un conjunto de semillas. Las consideraciones realizadas a propósito de los originales de un modelo también son válidas para las semillas. Las semillas utilizadas por un constructor sintético no son físicas. La obtención de los originales seguiría un proceso inverso al descrito para los modelos analíticos: el modelizador *imagina* una particularización de las características del modelo para obtener unos originales abstractos y a partir de éstos genera una imagen mental de un libro con esas características que sería el equivalente al original percibido sólo que en este caso ha sido concebido mentalmente.

Llamaremos *originales concebidos* a estos originales (en contraposición a los originales percibidos resultantes de un proceso analítico).

Finalmente el original concebido puede llegar a ser un original físico observable si el modelizador utiliza un dispositivo equivalente al sensor para el caso del

análisis, y que lo llamaremos *actuator*[1].

El proceso descrito plantea algunos interrogantes: ¿cómo es posible que los originales concebidos puedan producir imágenes mentales equivalentes a las de los originales percibidos en un modelo analítico? ¿Se está incumpliendo la regla de que el modelo es más simple que el original? Estas cuestiones se las han planteado algunos autores quienes pensaron que mientras que un original de un modelo descriptivo es más complejo que su modelo, no ocurre lo mismo con el original de un modelo prescriptivo (ver por ejemplo (Gonzalez-Perez and Henderson-Sellers, 2007)) donde parece que el modelo es más complejo que los originales. Sin embargo, nosotros no compartimos esa opinión.

La respuesta a esta aparente contradicción radica en que en el caso de los modelos sintéticos es fácil confundir el original abstracto con el original (concebido) del modelo. Volviendo al ejemplo anterior, lo que el modelizador obtiene como original concebido a través de un constructor sintético no son dos simples valores (las cadenas de texto correspondientes al título y el autor del libro) sino la imagen de un libro (que efectivamente tiene esas dos características) pero que se asemeja a la imagen que obtendría si hubiese percibido un libro físico. La clave para comprender cómo es posible la generación de esa imagen está en las semillas. Las semillas proporcionan no sólo los valores concretos necesarios para obtener el original abstracto sino también toda aquella información adicional para que lo que se obtenga sea realmente un libro y no simplemente un par de valores (original abstracto).

Quizás quede más claro si pensamos en el caso del modelo de mesa: con las sentencias y unos valores específicos para cada característica (4 patas, una superficie concreta, etc.) el modelizador debe ser capaz de *construir* (más bien, idear o imaginar) una mesa concreta siguiendo las instrucciones de un constructor sintético. Por esa razón, en ocasiones se piensa en los modelos sintéticos como los planos para construir algo, a lo que nosotros añadiríamos que las semillas son los materiales de construcción. La mesa sintetizada (no olvidemos que no es física) podrá tener otras características no relevantes para nuestro modelo (por ej. el color proporcionado por las patas elegidas) y es eso lo que permite afirmar que ese original concebido es más complejo que el modelo o dicho de otra forma, que se sigue manteniendo la regla de que el modelo es una simplificación del original.

[1]Para evitar confusiones mantendremos el término en inglés dado que no existe un término en español con un significado claramente equivalente

Lo anterior proporciona una explicación para cuestión planteada por Gonzalez-Perez and Henderson-Sellers (2007):

> We can observe this by realising that the main reason for creating backward-looking models is to get rid of unnecessary detail so that the resulting model is simpler than (but, hopefully, homomorphically equivalent to) the real thing; on the other hand, the main reason why we create forward-looking models is to explicitly specify as much detail as possible about an SUS that we intend to create. Detail is actively removed when looking back, but actively added when looking forward.We may assume that the nature of these two kinds of details is not the same. This issue, together with the mechanisms by which the back-to-forward shift happens, are certainly interesting topics by themselves, and will be considered as the theme for further research.

Efectivamente la naturaleza de los detalles activamente añadidos en los modelos sintéticos (que Gonzalez-Perez and Henderson-Sellers denominan forward-looking models) es muy diferente puesto que no provienen de la adición de nuevas (e irrelevantes) sentencias al modelo sino de las características de las semillas proporcionadas a un constructor sintético para obtener los originales concebidos.

El último caso es el de los modelos analógicos analíticos o completos. Como indicamos en el apartado 6.5.1, los originales físicos del modelo no son observables y por tanto el modelizador no puede obtener originales percibidos como en un proceso analítico. Sin embargo, el modelizador sí obtiene los originales del modelo por analogía con un modelo de referencia. Esos originales son equivalentes a los originales percibidos o concebidos (e igualmente imágenes mentales) y son los originales del modelo.

Llamaremos a estos originales *originales análogos* (en contraposición a los originales percibidos y concebidos resultantes de un proceso analítico o sintético respectivamente).

En definitiva, los originales de un modelo se corresponden con los que hemos denominado *originales percibidos* (para el caso de los analíticos), *originales concebidos* (para el caso de sintéticos) y *originales análogos* (para el caso de los analógicos analíticos o completos). Estos originales, que son los verdaderos originales del modelo, no deben confundirse, tal como hemos explicado, con los originales observables y los originales abstractos.

6.7.2. Modelos físicos y abstractos

Las consideraciones sobre originales respecto a su carácter físico o abstracto también pueden estar referidas a los modelos. ¿Existen modelos físicos? Desde nuestro punto de vista ya hemos indicado desde las propias definiciones de modelo que son entidades no físicas. La confusión puede venir desde diferentes enfoques:

- Se confunde modelo con la entidad física donde reside. Aunque los modelos no son físicos, residen en entidades físicas para que puedan ser útiles, es decir para que puedan ser aplicados la función de validación y los constructores analítico, sintético o analógico. Hemos denominado modelizadores a esas entidades físicas, que normalmente son humanos. No obstante, como describiremos en el Capitulo 9, los modelos pueden ser comunicados de un modelizador a otra persona, a una máquina (sistema informático) o de una máquina a otra, bajo ciertas condiciones. La tabla 6.1 muestra la correspondencia entre los principales términos de modelización para ambos tipos de modelizadores.
- Se confunde modelo con un prototipo físico del modelo. En la sección 6.2.1 definimos prototipo como un original representativo del modelo. Un prototipo físico observable sería la correspondiente entidad física asociada al prototipo. En el lenguaje natural, el término modelo suele usarse para designar a este objeto físico que o bien es objeto existente que sirve como muestra en un modelo analítico o es un objeto físico obtenido a partir de un original concebido en un modelo sintético.
- Se confunde modelo con una expresión del mismo. Tal es es el caso cuando por ejemplo un diagrama UML o la representación visual de una maqueta de un edificio. En el capítulo 8 trataremos en profundidad las relaciones entre modelos y lenguajes.

	Humano	**Sistema informático**
Modelo	Concepto	Estado ejecución (runtime)
Constructor	Proceso mental	Programa en ejecución
Original	Imagen mental	Estado de memoria digital
Sensor	Sentidos	Sensores electrónicos
Actuator	Acciones humanas	Dispositivos mecánicos o electrónicos

Tabla 6.1: Correspondencia entre modelo humano y mecánico

Como conclusión podemos afirmar que aunque en un lenguaje informal es habitual referirse a los modelos como algo físico, en el contexto de la teoría de la modelización es incorrecto hablar de modelos físicos.

6.7.3. El problema de la identidad

La distinción entre los distintos tipos de originales también permite abordar otra de las cuestiones confusas relacionadas con la modelización. Se trata del problema de la identidad: ¿cuándo podemos decir que dos originales son iguales?

Para poder responder a esta pregunta es necesario saber con claridad a qué tipo de originales nos estamos refiriendo. Fijémonos en el ejemplo de la figura 6.6. Si nos fijásemos en los originales físicos observables, podríamos tener por ejemplo varios libros físicos de El Quijote. Cada uno de ellos serían diferentes entre sí puesto que tienen una existencia material en tiempo y espacio diferenciada. Sin embargo, si pasamos al plano de los originales percibidos entonces diferentes libros físicos del Quijote podrían dar lugar al mismo original (percibido) si tienen las mismas características observables (pensemos por ejemplo en diferentes ejemplares de la misma editorial). Sin embargo, ejemplares del Quijote de diferentes editoriales (con diferente tamaño, color, etc.) también serían originales percibidos diferentes. Ahora pasemos al plano de los originales abstractos. Como hemos explicado, el original abstracto es el resultado de un proceso de abstracción y reducción, es decir, con el punto de vista del modelo sólo se consideran un conjunto limitado de características (en el ejemplo título y autor). Por tanto, diferentes originales percibidos (ejemplares del Quijote de diferentes editoriales), darán lugar al mismo original abstracto, puesto que en ese caso las características del modelo (título y autor) son las mismas. Es el modelo el que determina cómo diferenciar un original de otro. Así si el modelizador entendiera que libros de diferentes editoriales con el mismo título y autor deberían ser originales abstractos diferentes, entonces tendría que considerar alguna característica diferenciadora adicional, como el código ISBN para que pudieran ser tratados como diferentes. En definitiva, diferentes originales físicos observables pueden dar lugar al mismo original (percibido) y diferentes originales (percibidos) pueden dar lugar al mismo original abstracto.

Si observamos este problema desde la óptica de los modelos sintéticos, veremos que el razonamiento es el mismo: a partir de un modelo sintético se puede obtener un original abstracto que en virtud de los detalles aportados por diferentes

semillas podría dar lugar a diferentes originales (concebidos). A su vez un mismo original (concebido) podría dar lugar a diferentes originales físicos, observables una vez materializados mediante un actuator.

6.7.4. Modelos unitarios (singleton)

En el ejemplo del modelo de Sol hicimos referencia a un tipo de modelos que denominamos unitarios o singleton. Un modelo unitario no es sino un tipo especial de modelo que tiene un único original (recuérdese que el original de un modelo es el percibido, concebido o análogo). Como veremos a continuación, la confusión producida por no distinguir entre los tipos de originales descritos puede ser aún mayor cuando se trata con los modelos unitarios.

El origen del problema es confundir el modelo con el original abstracto, ya que existe una correspondencia completa (isomorfismo) uno-a-uno entre las características del modelo y sus valores concretos en el original abstracto. El objetivo de este apartado será ayudar a encontrar el origen de la confusión y clarificar ambos términos a la luz de lo descrito sobre los tipos de originales.

Es cierto que un original abstracto es una abstracción relacionada con un original físico observable y un original percibido, concebido o análogo. Igualmente un modelo, o más propiamente dicho un protomodelo, también es una abstracción en el mismo sentido. Sin embargo, de nuestras definiciones de modelo podemos deducir que un original abstracto no puede ser un modelo, ni siquiera en el caso en el que sólo exista un original. Un original abstracto es sólo un conjunto de valores concretos de las características que el modelizador ha considerado relevantes para el modelo. Sin embargo, si el original abstracto fuese un modelo no tendría posibilidad de ser usado, puesto que carece de la función de validación y los constructores que son precisamente lo que caracteriza a un modelo y hace que sea útil.

Partiendo del ejemplo representado en la figura 6.6 veamos otro ejemplo que puede servirnos para ilustrar el concepto de modelo unitario. Imaginemos que nuestro objetivo es obtener un modelo unitario (analítico) del libro Don Quijote. Al ser analítico presuponemos que el punto de partida es un libro físico observable del Quijote a partir del cual obtendríamos, tal como hemos explicado, un original percibido y un original abstracto que serían idénticos respectivamente a los indicados en el ejemplo para el modelo de libro. Sin embargo, el modelo unitario, construido usando un $\mathcal{AC}_m$ similar sería algo diferente (utilizamos nuevamente el

lenguaje SQL para ilustrar las sentencias del modelo):

```
create table ModeloLibroQuijote (
title varchar(9) primary key check (title='D.Quijote'),
isbn char(13) check (isbn='84-9714-056-7',
author char(9) check (author='Cervantes'))
```

La modificación del conjunto de sentencias (expresadas en SQL mediante la adición de algunas restricciones), ha permitido al modelizador crear un modelo que tiene sólo un original, sobre todo porque indica una característica que es única para un tipo de ejemplar de una editorial concreta. Aunque puede haber millones de ejemplares físicos *iguales*, el original percibido sería el mismo para todos ellos y por tanto su modelo sería unitario. Podría parecer a primera vista que los modelos unitarios sólo podrían ser analíticos puesto que si tuviésemos un constructor sintético podríamos generar más originales y por tanto dejaría de ser un modelo unitario. Sin embargo, en virtud de lo expuesto en el apartado 6.7.3, será un modelo unitario si cada constructor sintético tiene restringida la creación de originales a partir de un conjunto único de semillas, que es el caso del ejemplo del modelo de libro del Quijote que acabamos de exponer. Finalmente cabe preguntarse si un modelo unitario podría ser completo. La respuesta es que sí es posible. Veámoslo con otro ejemplo:

Imaginemos el modelo de una persona para realizar una estatua de cera de la misma. Inicialmente se trata de un modelo analítico que está basado en un original físico observable (una persona concreta). El original percibido es, tal como hemos explicado, la imagen mental que capta el modelizador de esa persona, por ejemplo visualmente y a partir del cual obtiene el protomodelo mediante el constructor analítico del modelo. El protomodelo describe a la persona con el nivel de detalle necesario para poder hacer la estatua. Con este conocimiento, el modelizador añade un constructor sintético al modelo con el objetivo de obtener un original concebido, y posteriormente físico observable en el nivel de abstracción recogido en el modelo (sólo su aspecto visual), por ejemplo una estatua de cera cuyo aspecto y por tanto original percibido es idéntico al de la persona real. Es interesante que el resultado conseguido permite que el mismo original percibido pueda ser generado por síntesis y aunque los originales físicos observables sean diferentes, el original (percibido y concebido) del modelo es el mismo, en virtud de lo explicado en el apartado 6.7.3.

Diversos autores han tratado este problema y dado diferentes nombres a los

modelos unitarios. Por ejemplo Kühne (2006a) los denomina *token-models* (en contraposición a *type-models)*, Hesse (2006) *feature projection models* (frente a *placeholder projection models*) y Gonzalez-Perez and Henderson-Sellers (2007) emplea el término *isotypical mapping* (para distinguirlo de *prototypical/metatypical mappings*). Alguno de estos autores (por ejemplo Kühne (2006a)) analizan las características de este tipo de modelos y concluyen que tienen diferentes características respecto al resto de los modelos. En nuestra opinión, tal diferenciación no existe puesto que como hemos explicado, los modelos unitarios son modelos estándar con la única particularidad de que su dominio está formado únicamente por un original (que podrá ser percibido, concebido o análogo).

6.8. Conclusiones y trabajo relacionado

Tal como habíamos planteado en la introducción de este capítulo nuestro objetivo era abordar una teoría de la modelización basada en el método científico que clarificara muchos de los aspectos confusos que existen actualmente sobre esta materia. La Definición 6.7 resume justamente ese enfoque al incluir tres de las actividades típicas del método científico que describimos en el capítulo 5: análisis, síntesis y validación. Adicionalmente la Definición 6.8 incluye la analogía como otro de los aspectos que también juegan un papel importante en el método científico. A pesar de la novedad de este enfoque y como nos propusimos en la introducción, nuestra definición de modelo en efecto subsume la esencia de las concepciones de modelo encontradas en la literatura aunque con algunos matices interesantes, siendo el más relevante la diferenciación entre modelo y protomodelo. Consideramos como uno de los pilares principales de la teoría el que un modelo no es simplemente un conjunto de sentencias sobre algo tal como afirman Seidewitz y Gonzalez-Perez and Henderson-Sellers aunque coincidimos con estos autores en la importancia de las sentencias en la definición de modelo diferenciando claramente entre el conjunto de sentencias, que denominamos protomodelo, del verdadero modelo que para serlo ha de incluir alguno de los constructores que permite que un modelo sea útil. En definitiva, en nuestra opinión si un modelo no incluye mecanismos para analizar o sintetizar no puede ser realmente un modelo.

En nuestra exposición hemos ido profundizando en los conceptos asociados a la definición de protomodelo: concepto, sentencia, abstracción, etc. al considerar que para que la definición de protomodelo (y posteriormente la de modelo) tenga un

mejor grado de precisión era necesario saber a qué nos estábamos refiriendo, aún siendo conscientes que abordar estas cuestiones implican adentrarse en terrenos que están en el límite del ámbito de este estudio, tales como la filosofía o la semiótica y que han dado lugar a debates a lo largo del tiempo en diversas épocas e incluso algunos de ellos continúan en el momento actual.

El siguiente paso, tomando como referencia el método científico, ha ofrecido una clasificación de los modelos en cuatro tipos básicos: analíticos, sintéticos, analógicos y completos. Esta clasificación define con mayor precisión lo que la literatura denomina con diversos nombres como backward/descriptive y forward/prescriptive y que sólo abarcan a los modelos analíticos y sintéticos respectivamente. Hemos llevado hasta sus últimas consecuencias el principio de verdad enunciado por Favre por el que lo que diferencia a los modelos descriptivos de los prescriptivos es identificar dónde está *la verdad*, o dicho de otro modo, cuál existió primero.

Una de las conclusiones interesantes ha sido el de la existencia de los modelos completos, que no suelen ser mencionados en la literatura, y en los que es interesante observar como en su dominio coexisten dos tipos de originales: aquellos que existían antes del modelo (y que el constructor analítico toma un subconjunto de éstos como base para obtener el protomodelo) y los que son generados posteriormente mediante un constructor sintético. En este tipo de modelos se produce una vinculación estrecha entre original y modelo para preservar el principio de verdad: si el modelo cambia es porque ha habido un cambio en los originales preexistentes y el constructor analítico debe reflejar ese cambio para actualizar en su caso el protomodelo, pero al mismo tiempo ese cambio afectará a los originales puesto que éstos deben adaptarse al modelo a través de cada constructor sintético que también se verá afectado. Podemos concluir por tanto, que en los modelos completos los cambios deben ser síncronos y afectan a modelo y a todos sus originales por igual.

Otra cuestión interesante es la planteada por algunos autores (véase por ejemplo (Bézivin, 2005; Gonzalez-Perez and Henderson-Sellers, 2007; Kühne, 2006b) al afirmar que el carácter descriptivo (analítico) o prescriptivo (sintético) de los modelos es en realidad un rol y por consiguiente en realidad no existen modelos que sean intrínsecamente analíticos o sintéticos. Compartiríamos esa opinión si estuviera referida a protomodelos, puesto que un protomodelo no incluye de manera intrínseca un carácter descriptivo o prescriptivo. Sin embargo, cuando un

modelizador crea un modelo lo realiza siguiendo alguno de los procesos descritos para los cuatro tipos de modelos expuestos. Esto no es obstáculo para que puedan ocurrir conversiones de tal manera que a partir de un modelo de un determinado tipo pueda evolucionarse hacia un modelo de otro tipo dando lugar a un nuevo modelo cuya finalidad y comportamiento serán diferentes. Dicho de otro modo, para nosotros dos modelos con el mismo protomodelo siendo uno analítico y el otro sintético son dos modelos diferentes puesto que el modelo es el conjunto de protomodelo y constructores, no únicamente su protomodelo.

Parece claro, e incluso habitual, tal como hemos mostrado con diversos ejemplos, que un modelo analítico puede pasar a ser completo en una fase posterior si el modelizador decide añadir un constructor sintético. Esto es cierto incluso para los modelos unitarios tal como hemos mostrado con el ejemplo de modelo de una persona para realizar una estatua de cera. El caso contrario, aunque probablemente menos habitual también es posible: un modelo sintético puede pasar a ser completo si el modelizador descubre posteriormente tras un proceso de análisis que su modelo también describe entidades que existían previamente aunque no pudo utilizarlas para la creación del modelo al haber desconocido su existencia. Una vez incorporado el constructor analítico el modelizador enriquece el modelo e incluso puede mejorarlo (perfeccionar sus sentencias y cada constructor sintético) gracias al proceso analítico realizado.

La cuestión sobre las dependencias entre original y modelo también han sido tratadas por la literatura y dan lugar nuevamente a preguntas sobre el verdadero significado de modelo. En primer lugar parece claro que original y modelo son conceptos interdependientes, puesto que uno no puede existir sin el otro. En efecto, para que una entidad pueda ser considerada un modelo debe tener al menos un original con el que esté relacionado. Esto se deduce fácilmente de nuestras definiciones puesto que un constructor analítico no puede existir sin una muestra y un constructor sintético tampoco si no es capaz de sintetizar un original. En ambos, casos el nexo final de unión entre original y modelo es el homomorfismo que subyace tras las sentencias del modelo. Sin embargo, esa dependencia no quiere decir que una *misma* entidad (se entiende que no física) pueda ser original o modelo según como sea usada, o como apunta Favre (2004a) que la noción de modelo sea relativa. Bézivin (2005) va mucho más allá al afirmar *everything is a model*. No podemos compartir ese punto de vista. En nuestra opinión, tal como deja bien claro la sección 6.7 lo único que tienen en común original y modelo

es su naturaleza no física, pero son conceptos muy diferentes. Mientras que el original es algo estático el modelo es su contrapartida abstracta y dinámica ya que cuenta con constructores analíticos, sintéticos o analógicos. El único caso en el que original y modelo se refieren a lo mismo es cuando el dominio de un modelo es un conjunto de modelos, o metamodelización, que será tratado en el capítulo 7. Sin embargo, incluso en ese caso, los originales (modelos) son tratados como algo estático (percibido, concebido o análogo). Por el contrario lo que sí podemos afirmar es que cualquier entidad física o no, puede servir de base directa o indirectamente (como en el caso de modelos analógicos) para obtener un modelo.

Capítulo 7

Relaciones entre modelos

En el capítulo anterior se han descrito los fundamentos de la teoría propuesta. En nuestra opinión, los conceptos tratados, todos ellos centrados en el significado de modelo, constituyen la base para comprender y utilizar la modelización de una manera básica en cualquier área de conocimiento.

En este capítulo nos centraremos en conceptos secundarios de modelización, pero no por ello menos importantes en la práctica. Nuestra intención es abordar otras cuestiones que han sido intensamente objeto de debate en la literatura y que ya han sido tratados en el capítulo 4, pero desde la perspectiva de la teoría desarrollada hasta el momento. El primer tema es el de la herencia, que abordaremos desde un punto de vista más genérico, definiendo el concepto de submodelización. A continuación trataremos el importante tema de la metamodelización. Continuaremos con la introducción de una relación entre modelos que se deriva de las características de un modelo y que dará lugar a un nuevo tipo de modelos denominados modelos característicos. Finalmente, abordaremos la transformación de modelos como un tipo de relación entre modelos más general, descrita desde la perspectiva de nuestra teoría.

7.1. Submodelos

En la sección 4.4.2 se trataron los principales problemas que se plantean sobre herencia desde el punto de vista de los estándares OMG. A continuación abordaremos un enfoque basado en nuestra propuesta desde una visión general.

Definición **7.1.** Submodelo
Dados, $m, m' \in \mathcal{M}$, se dice que m es un **submodelo** de m' y lo denotamos

como $m \triangleright m' \iff \mathcal{O}_m \subseteq \mathcal{O}_{m'}$ siendo $\mathcal{M}$ el conjunto de todos los modelos y $\mathcal{O}$ el conjunto de originales de un modelo. Así mismo, se dice que m' es un **supermodelo** de m.

Un ejemplo simple es el de modelo de mobiliario m' que es supermodelo del modelo de mesa m. En efecto, todos los originales del modelo de mesa son también originales del modelo de mobiliario. Además todo original del modelo de mesa cumple la función de validación de los modelos de mesa y mobiliario mientras que un original de mobiliario cualquiera cumple la de su modelo pero no la del modelo de mesa.

La anterior definición también indica, si nos fijamos en las funciones de validación, que todo aquello que sea cierto para cada original de un supermodelo debe ser también cierto para cada original de sus submodelos. Sin embargo, eso no significa necesariamente que deba existir inclusión de sentencias (que las sentencias del supermodelo estén incluidas en las del submodelo).

7.1.1. Comparación de submodelización y herencia

Obsérvese que lo anterior no significa que las sentencias del modelo m' ($\mathcal{D}_{m'}$) tengan que estar incluidas necesariamente en el conjunto de sentencias del submodelo m ($\mathcal{D}_m$) puesto que la validación de uno y otro modelo puede estar construida de forma diferente. De hecho, la Definición 7.1 sólo requiere que el conjunto de originales del submodelo esté incluido en el del modelo y no indica en ningún momento qué condiciones deben satisfacer las sentencias o los constructores de ambos modelos. No obstante, cualquier condición que pudiera existir entre los elementos de estos modelos conduciría a casos particulares de submodelización.

Podemos encontrar ejemplos en los que la verificación de una función de validación implique la de otra función de validación sin que exista inclusión de sentencias entre ambas, aunque no sean habitualmente interpretados de este modo. Son casos en los que las sentencias del submodelo no incluyen las del supermodelo porque son innecesarias o superfluas, mientras que estas características sí están implícitas en los originales. Considérese por ejemplo el caso del modelo de las personas que pagan impuestos (supermodelo) y el modelo de los consumidores que es un submodelo puesto que todos los consumidores pagan impuestos indirectos. O el ejemplo de los mamíferos y los jugadores de fútbol: lógicamente todo jugador de fútbol es un mamífero pero no es habitual que un modelizador incluya esa característica en el modelo analítico de los jugadores de fútbol.

Así por ejemplo, respecto a los constructores, pueden existir casos en los que $\mathcal{SC}_m$ y $\mathcal{SC}_{m'}$ sean procesos independientes y también aquellos casos en los que $\mathcal{SC}_{m'}$ sea usado por $\mathcal{SC}_m$, por ejemplo las llamadas encadenadas a los constructores de las superclases en las clases heredadas en un lenguaje orientado a objetos como Java o C++. $\mathcal{AC}_m$ y $\mathcal{AC}_{m'}$ también podrían estar relacionados de manera parecida, por ejemplo cuando las peculiaridades del $\mathcal{AC}_{m'}$ incluyen las de $\mathcal{AC}_m$.

Sin embargo, el caso más habitual en la práctica es cuando las sentencias del supermodelo m' son un subconjunto de las sentencias del submodelo m. Éste sería el caso de la herencia tal como es definida tradicionalmente. Dicho de otra forma, las sentencias de m están formadas por la unión de dos conjuntos: el conjunto de las sentencias del supermodelo m' más otro conjunto de sentencias que podrían ser consideradas como las sentencias específicas del submodelo m. En ese caso, se dice que esas sentencias específicas del submodelo refinan o amplían las sentencias del supermodelo, del mismo modo que en la herencia una subclase refina o amplía los atributos y/o métodos de una superclase.

7.1.2. Propiedades de submodelo

Puesto que la relación entre submodelo y supermodelo de la Definición 7.1 está definida mediante la relación de inclusión de conjuntos, podemos deducir algunas de sus propiedades:

1. **Reflexiva**: $m \triangleright m$ puesto que $\mathcal{O}_m \subseteq \mathcal{O}_m$
2. **Transitiva**: Si $m \triangleright m'$ y $m' \triangleright m''$ entonces $m \triangleright m''$, ya que si $\mathcal{O}_m \subseteq \mathcal{O}_{m'}$ y $\mathcal{O}_{m'} \subseteq \mathcal{O}_{m''}$ entonces $\mathcal{O}_m \subseteq \mathcal{O}_{m''}$

En el contexto de los estándares OMG sobre modelización, se utiliza profusamente el concepto de herencia dada su fundamentación en la orientación a objetos. Algunos autores (ver por ejemplo (Kühne, 2006a)) han resaltado que la herencia tiene la propiedad transitiva e incluso sugieren que tal propiedad puede usarse para comprobar si una determinada relación es de herencia, dados los problemas que en ocasiones se plantean para distinguir la herencia de otras relaciones como la de metamodelización y que fueron descritas en la sección 4.4.2. La submodelización que hemos descrito es más general y la transitividad es una de sus propiedades necesarias pero no suficientes, es decir, si una relación es transitiva no es suficiente para afirmar que sea de submodelización. Por ejemplo, si tenemos una relación entre modelos que no sabemos si es de metamodelización (que trataremos en la

sección siguiente) o de submodelización y sabemos que es transitiva sólo podríamos decir que *puede* ser de submodelización (puesto que podría ser cualquier otra relación, incluida la de metamodelización en caso de que *no fuese* antitransitiva), mientras que si no es transitiva sí podemos afirmar con certeza que no es de submodelización. Para poder asegurarlo con certeza deberíamos hacer uso de la Definición 7.1 puesto que la inclusión de conjuntos de originales sí es una condición necesaria y suficiente para que la relación sea de submodelización. Nótese que el que una relación *no sea antitransitiva* no significa que *sea transitiva*. Como conclusión podemos decir que el conocimiento de las propiedades entre modelos ayuda a evitar confusiones entre ellas, aunque en general no permiten determinar de qué relación se trata a menos que se apliquen las definiciones proporcionadas.

7.1.3. Equivalencia entre modelos

En general la relación de submodelización no es simétrica. En efecto, no puede afirmarse que sea simétrica puesto que si $m \triangleright m'$ no implica que $m' \triangleright m$. Sin embargo, podemos definir una relación de equivalencia entre modelos para la que sí se cumple la condición de simetría:

Definición **7.2.** Equivalencia de modelos
Sean dos modelos m_1 y m_2, diremos que son **equivalentes** y lo denotaremos como $m_1 \sim m_2$ si se cumple que $m \triangleright m'$ y $m' \triangleright m$

Definición **7.3.** Clase de modelos equivalentes
Dado un modelo m denominaremos **clase de equivalencia** de m al conjunto $[m] = \{n \in \mathcal{M} | n \sim m\}$

La relación $\sim$ es efectivamente una relación de equivalencia puesto que tiene las propiedades reflexiva, transitiva y simétrica, tal como puede deducirse de manera trivial al cumplirse que si $m_1 \sim m_2$ entonces $\mathcal{O}_m = \mathcal{O}_{m'}$.

Con el fin de clarificar el concepto de equivalencia de modelos es conveniente realizar algunas observaciones:

- Modelos analíticos, sintéticos, analógicos o completos, con el mismo conjunto de originales son equivalentes. Esto quiere decir que para que dos modelos sean equivalentes no importa cómo se han obtenido sus funciones de validación, ni sus constructores sino sólo que compartan el mismo conjunto de originales. Dicho de otro modo, la relación de equivalencia se define a nivel de protomodelo.

- Dos modelos equivalentes tienen funciones de validación iguales, pero eso no implica que tengan el mismo conjunto de sentencias.
- Dos modelos que fuesen expresados en lenguajes diferentes (sus protomodelos y/o sus constructores) podrían ser equivalentes. La equivalencia de modelos está relacionada con los vértices de *conceptualización* y *realidad* del Triángulo del significado pero no con el de *lenguaje*.

7.2. Metamodelos

En esta sección trataremos la metamodelización que es uno de los temas más debatidos en la literatura sobre modelización. En general, nuestro punto de vista inicial es que en esencia un metamodelo no es otra cosa que un modelo cuyos originales también son modelos. Este sencillo enfoque contrasta con el de aquellos autores quienes sitúan a la metamodelización en el centro de sus teorías de modelización (Bézivin, 2001; Gašević et al., 2007; Gitzel and Hildenbrand, 2005; Gitzel and Korthaus, 2004; Mellor et al., 2004; Terrasse et al., 2006) o de aquellos que identifican metamodelo con el modelo de un lenguaje de modelización (Aß mann et al., 2006; Favre, 2005; Gonzalez-Perez and Henderson-Sellers, 2007; Kurtev, 2007; Ober and Prinz, 2006; OMG, 2002; Seidewitz, 2003) (ver RCM de Metamodelo en sección 4.1.2).

Definición **7.4.** Metamodelo
Se dice que un modelo mm es un **metamodelo** $\iff \forall o \in \mathcal{O}_{mm}, o \in \mathcal{M}$. Denominamos **metaoriginales** a los originales de los modelos pertenecientes a $\mathcal{O}_{mm}$. Alternativamente, en virtud de la Definición 6.10, si mm es un metamodelo de m, la **relación de metamodelización** se expresará mediante $mm \models m$, simplemente como un caso particular de modelización.

No es difícil encontrar ejemplos de metamodelos incluso en la vida diaria. Considérese el caso de un fabricante de automóviles.

Este fabricante tiene un conjunto de normas que establecen cómo ha de construirse cada tipo de automóvil (diremos que esas normas describen cómo son los modelos sintéticos de coche y cómo crearlos). Seguramente, el fabricante tiene un conjunto de normas sobre cómo han de construirse nuevos coches. Esas normas describen cómo son los modelos sintéticos para los nuevos coches y cómo crearlos. La experiencia acumulada a lo largo de los años ha asesorado a los ingenieros de la compañía sobre cómo definir esas normas de desarrollo (el contructor analítico

del metamodelo). Estas normas no son rígidas, sino que se refinan continuamente basándose en la experiencia (positiva o negativa) adquirida por la compañía (el constructor analítico refina continuamente el metamodelo). Las normas especifican los componentes de un coche y como diseñarlos, qué planos se necesitan y como construirlos, cuáles son los materiales más adecuados que han de utilizar, cómo construir de una manera ecológica, etc. En definitiva, se trata de un metamodelo que especifica cómo hacer nuevos modelos de coches que los ingenieros empleados por la compañía utilizan para el desarrollo de cada nuevo modelo de coche que deseen sacar al mercado.

También podemos encontrar habitualmente metamodelos en el ámbito de la ingeniería del software. Consideremos por ejemplo el modelo Entidad -Relación (Chen, 1976), el cual, a pesar de llevar la palabra *model* en su nombre en lugar de la palabra *metamodel*, es considerado comúnmente como un metamodelo puesto que sirve para generar modelos Entidad - Relación.

Obsérvese que la Definición 7.4 puede ser aplicada de forma iterativa, es decir, dados los modelos mmm, mm y m, podría ocurrir que $mmm \models mm$ y que $mm \models m$. Se diría entonces que mmm es un **meta-metamodelo**. Así, continuando con el ejemplo anterior, considérese que una hipotética asociación de empresas del sector del automóvil propusiese un conjunto de recomendaciones para el desarrollo de guías o especificaciones sobre cómo desarrollar nuevos modelos de coche y que deberían ser adoptadas por todas las empresas de dicha asociación. Podríamos seguir aplicando la definición de metamodelo a un cuarto modelo mmmm que fuese metamodelo de mmm, que llamaríamos metametametamodelo y así sucesivamente. En la literatura podemos encontrar que tal sucesión de metamodelos se estudia utilizando el concepto de grado de abstracción de un metamodelo, dando lugar a diferentes teorías de modelización basadas en niveles: por ejemplo los niveles M0 a M3 propuestos por OMG (ver figura 4.16 tomada de (OMG, 2011b)) o los niveles EIA/CDIF (Flatscher, 2002) que ya fueron mencionados en la sección 4.3. Nosotros no estamos interesados en el estudio de la metamodelización de un grado mayor que uno puesto que no creemos que aporten nada nuevo a nuestra teoría de la modelización que no haya sido mostrado a través de la simple relación de metamodelización. La figura 7.1 muestra un ejemplo de metamodelo completo cuyos originales (modelos) también son modelos completos y en el que está resaltado (1) el proceso de construcción del metamodelo mediante su $\mathcal{AC}_{mm}$, (2) la creación de originales (modelos) mediante su constructor sintético $\mathcal{SC}_{mm}$ y

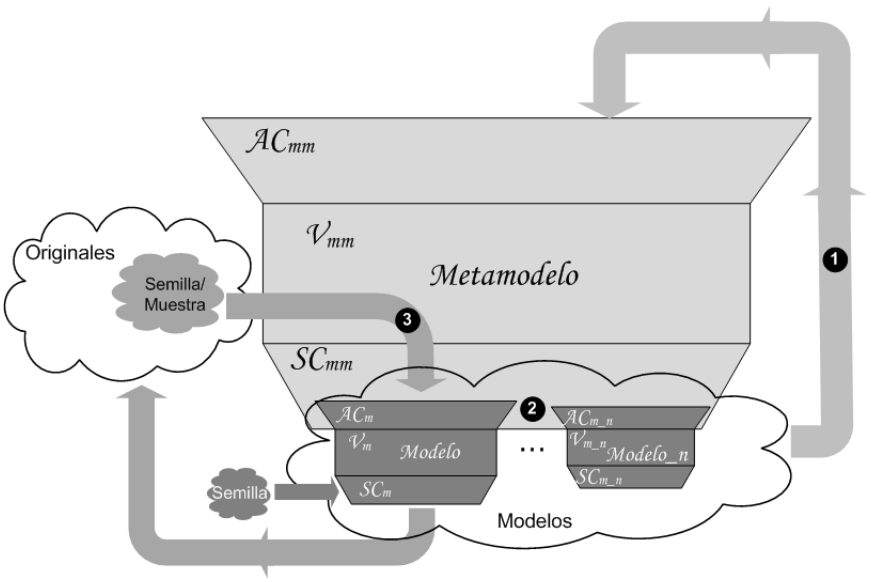


Figura 7.1: Metamodelos

(3) el constructor analítico $\mathcal{AC}_m$ de sus modelos cuya relación con el $\mathcal{SC}_{mm}$ será descrita en el próximo apartado.

7.2.1. Elementos de metamodelo

Como hemos explicado hasta el momento, la noción de metamodelo no es a priori más compleja que la de cualquier otro modelo. Nuestra intención no es reclamar originalidad en esa idea de simplicidad que es compartida, al menos a la hora de expresarlo, con otras referencias (véase por ejemplo (Bézivin, 2005; Favre, 2005; OMG, 2003)). Por el contrario, tal como veremos a continuación, nuestro enfoque permite profundizar en el verdadero significado de metamodelo. Así mismo, aporta a la teoría de la modelización algunas conclusiones interesantes que se deducen de la teoría que hemos descrito hasta el momento, no sólo sobre el caso particular de los metamodelos, sino sobre cómo se generan los modelos en general. Para ello, vamos a profundizar en los elementos que constituyen un metamodelo como caso particular de modelo, es decir, sus sentencias y sus constructores, así como estudiar sus relaciones con los constructores de sus originales (modelos).

Protomodelo de metamodelo ($\mathcal{P}_{mm}$)

En virtud de la Definición 6.2, las sentencias de un modelo deben permitir determinar si cualquier entidad es un original del mismo, y ser lo suficientemente

específicas para que sus originales no se confundan con los de otros modelos. Por tanto, si un metamodelo mm tiene como original un modelo m, y éste tiene a su vez sentencias y constructores, entonces la primera conclusión es que las sentencias de mm deben *decir algo* sobre las sentencias y los constructores de m. Así, si por ejemplo m es analítico, las sentencias de mm deberían permitir validar si una determinada entidad tiene o no un $\mathcal{AC}_m$ que cumpla unas determinadas condiciones definidas por esas sentencias. Lo mismo podríamos decir de manera similar sobre el $\mathcal{AgC}_m$ de un modelo analógico. Por otro lado, si m fuese un modelo sintético las sentencias de mm deberían validar si cualquier entidad tiene algún $\mathcal{SC}_m$, qué condiciones cumplen esos $\mathcal{SC}_m$ y qué semillas de $\mathcal{SC}_m$ son válidas. Las sentencias de mm deben (1) hablar del constructor analítico de los modelos m, es decir, las sentencias deben describir o prescribir, como un modelizador analiza o debe analizar; (2) hablar del constructor analógico de m, si éstos fueran analógicos, es decir, cómo realiza el modelizador analogías con un modelo de referencia de m; y (3), en el caso de modelos sintéticos deberá hablar del constructor sintético, es decir, de cómo genera sus originales. En definitiva, el modelizador de m ha tenido que describir y/o especificar en las sentencias del protomodelo cómo el modelizador de m debe modelizar, en cualquiera de sus variantes analítica, sintética o analógica. Ésta es sin duda una característica que distingue a los metamodelos del resto de los modelos y que se deduce de las propias definiciones de modelo y metamodelo.

La anterior afirmación puede resultar sorprendente a cualquier ingeniero de software habituado a trabajar con metamodelos de ingeniería del software, sobre todo teniendo en cuenta que tal como hemos indicado desde el principio, lo común es que su concepción de modelo sea la que hemos denominado protomodelo (de un metamodelo en este caso) y por tanto nunca se haya planteado cómo es posible que un modelo original de un metamodelo pueda modelizar sin que el metamodelo permita validar cómo analiza o sintetiza ese modelo. Hay dos respuestas muy plausibles a esta cuestión:

1. El modelizador tiene que enfrentarse a numerosos problemas por tener que trabajar con un protomodelo en lugar de con un modelo. Es como si a un arquitecto se le proporcionase únicamente una guía para crear planos sin poder validar cuáles son los materiales que podrán usarse en los diferentes tipos de edificios (elección de semillas) o cuál es el procedimiento para construir un edificio a partir de un plano (constructor sintético).

2. Lo cierto es que el modelizador, a pesar de contar (en apariencia) únicamente con el protomodelo de metamodelo finalmente construye y usa modelos a partir de éste. Si lo hace, debe ser porque el metamodelo le especifica cómo hacerlo. Y de hecho, así es. Para facilitar la transmisión de su metamodelo, el modelizador de mm incluye ejemplos prototípicos en la descripción de las sentencias de su metamodelo. De esta manera, los futuros modelizadores, usando la técnica del constructor sintético por prototipo, descrita en el apartado 6.3.1, pueden emplearlos para reconstruir los constructores del metamodelo y, así, poder utilizar este último para sus propósitos. Nótese además que los ejemplos, que deben ser modelos, deben especificar entradas (dominios a ser modelizados) y salidas (modelos acordes al metamodelo). Es decir, los ejemplos transmitidos deben incluir una muestra de sus originales para que el futuro modelizador pueda reconstruir esos modelos prototípicos, usando esta vez el constructor analítico por prototipo (apartado 6.2.1). Este podría haber sido el caso del metamodelo de E/R descrito en anteriormente, en el que el modelizador (Chen) describe ejemplos prototípicos sobre cómo crear modelos que sean originales de su protomodelo.

   Otro ejemplo bien conocido es el (proto)metamodelo asociado a la sintaxis abstracta de UML especificado en (OMG, 2011b,c) donde se encuentran muchos ejemplos que ilustran las características esenciales de cada uno de los metamodelos de UML y cómo son usados para modelizar (entidad, atributo, relación, clase, actor, etc.). Sin la ayuda de estos ejemplos sería prácticamente imposible aplicar dicho metamodelo UML (aunque como hemos descrito con detalle en el capítulo 4 tal labor no está exenta de problemas debido a la complejidad e inconsistencias de dicho (proto)metamodelo).

   Obsérvese que al usar ejemplos y no especificar de manera precisa los elementos del protomodelo se da libertad a los modelizadores para analizar, sintetizar o realizar modelos análogos de formas diferentes. Ésta es la razón por la que diferentes modelizadores que usan el mismo metamodelo para modelizar el mismo dominio obtienen modelos que como mucho son equivalentes pero que habitualmente serán diferentes.

   Otras veces, los ejemplos no bastan para describir el constructor sintético de un metamodelo, y es necesario añadir una explicación que complemente la descripción de éste para que el futuro modelizador lo comprenda. Y por último, en otras ocasiones, otros metamodelos son especificados con

total precisión. A modo de ejemplo, podemos considerar un programa de ingeniería inversa utilizado para obtener un modelo conceptual a partir del estudio de un determinado esquema de base de datos relacional. El creador del programa tiene en mente un metamodelo de estos modelos conceptuales que se van a generar tan preciso que le permite desarrollar un programa informático que analiza de acuerdo con el metamodelo, generando modelos cuyo constructor analítico son las ejecuciones concretas del programa de ingeniería inversa.

Constructor analítico de metamodelo ($\mathcal{AC}_{mm}$)

El constructor analítico de un metamodelo $\mathcal{AC}_{mm}$ representado en la figura 7.1(1) no se distingue del constructor de cualquier modelo en general puesto que cubre el proceso por el que un modelizador construye un modelo a través de análisis tal como fue descrito en la sección 6.2. En este caso la muestra está formada por modelos existentes y el modelizador de mm los observa y analiza para enunciar las sentencias de mm. Estas sentencias deben permitir validar no sólo las características de los modelos de la muestra sino también sus constructores

Siguiendo con el ejemplo anterior sobre el fabricante de coches, sería el caso en el que el metamodelo de los modelos de automóvil se construyese *a posteriori* analizando las características comunes de los modelos existentes. Al tratarse de un metamodelo, el modelizador debe observar no sólo los planos de los modelos (protomodelos) de muestra sino también sus constructores. En el ejemplo, considerando que los modelos de automóvil son lógicamente sintéticos tendría que analizar sus $\mathcal{SC}_m$, es decir, cómo construir coches, incluyendo sentencias sobre el proceso productivo de esos coches.

Constructor sintético de metamodelo ($\mathcal{SC}_{mm}$)

Si el metamodelo mm fuese sintético, significa que existiría previamente a sus originales m. El modelizador de mm lo habría creado mediante un proceso sintético como fue descrito en la sección 6.3. Este podría ser el caso del modelo E/R en el que Chen ideó originalmente el metamodelo E/R.

A continuación analizaremos qué ocurre cuando los modelos generados por el $\mathcal{SC}_{mm}$ son analíticos, sintéticos/completos y analógicos:

- **Analíticos**: cuando el $\mathcal{SC}_{mm}$ genera modelos analíticos existe una relación

muy interesante de los constructores analíticos $\mathcal{AC}_m$ con el constructor sintético de mm $\mathcal{SC}_{mm}$. La intuición es clara. Cada vez que el modelizador de mm utiliza el $\mathcal{SC}_{mm}$ se crea un nuevo modelo analítico (su protomodelo y sus constructores). Una parte de ese proceso, aquella por la que se crean las sentencias del nuevo modelo analítico (su protomodelo), es por definición (ver Definicion 6.5) el $\mathcal{AC}_m$ de ese nuevo modelo. Por tanto, una ejecución de $\mathcal{SC}_{mm}$ parece estar relacionada con el $\mathcal{AC}_m$. Esta característica de los metamodelos está reflejada en el ejemplo de la figura 7.1(2) por el solapamiento entre el trapecio que representa $\mathcal{SC}_{mm}$ y el correspondiente a $\mathcal{AC}_m$. Por otro lado, la semilla que sirve como entrada al $\mathcal{SC}_{mm}$ también puede ser vista desde el punto de vista del proceso analítico del modelo resultante. Puesto que estamos suponiendo que m es analítico, sus originales $\mathcal{O}_m$ deben existir previamente. La intuición dicta nuevamente que una muestra de esos originales existentes tiene que tener alguna relación con la semilla a partir de la cual $\mathcal{SC}_{mm}$ genera m (ver 7.1(3)), puesto que si $\mathcal{AC}_m$ se basa en una muestra de $\mathcal{O}_m$ para obtener su protomodelo entonces $\mathcal{SC}_{mm}$ también debería basar su semilla en una muestra de $\mathcal{O}_m$ para conseguir el mismo objetivo, es decir, construir el modelo.

Aunque la intuición lo sugiere, aparentemente no podemos decir estrictamente que todos los modelos analíticos sean generados a través del constructor $\mathcal{SC}_{mm}$ de un metamodelo. Sin embargo, es posible ofrecer una explicación razonada que permita comprender a qué se deben las dos relaciones que acabamos de describir (entre $\mathcal{SC}_{mm}/\mathcal{AC}_m$ y entre semillas/originales) basándonos en uno de los principios en los que se basa nuestra teoría: el método científico.

En el apartado 5.2.1 describimos que una de las fases del método científico es la formulación de hipótesis, y que hay dos métodos para su obtención: el analítico y el sintético. Hasta el momento, en nuestra teoría hemos visto únicamente cómo, de manera similar, las hipótesis de un modelo se obtienen mediante análisis a través del constructor analítico, pero no hemos explicado ninguna aplicación del segundo método. Precisamente ese método es el que permite explicar la relación entre $\mathcal{SC}_{mm}$ y $\mathcal{AC}_m$: como indicamos en la sección 5.2.1 para obtener hipótesis el método sintético propone 3 fases: 1) Análisis, 2) Deducción y 3) Delimitación del ámbito de estudio. Consideremos que en vez de realizar un proceso analítico para obtener las hipótesis

de m, realizáramos un proceso sintético partiendo de las hipótesis de mm siguiendo esas fases:

1. **Análisis:** como indicamos en el caso del método científico, este análisis no tiene como objeto la observación de la realidad (muestra de originales) sino que su objeto son sentencias que no son otras que las sentencias del protomodelo de mm.
2. **Deducción:** mediante inferencia deductiva en la que las premisas son las sentencias del protomodelo de mm y las conclusiones las hipótesis del modelo.
3. **Delimitación del ámbito de estudio:** se determina cual es el ámbito de estudio en el que son de aplicación las conclusiones (hipótesis del modelo). En nuestro caso, el ámbito de estudio es el dominio de m y por tanto és de éste de dónde se obtiene la particularización de las sentencias que se acaban de deducir. En esta particularización es donde intervienen las semillas de $\mathcal{SC}_{mm}$.

A continuación, tal como se explica en la sección 5.2.1 se realiza la formulación de predicciones. Este paso se corresponde con la ejecución del constructor sintético del metamodelo $\mathcal{SC}_{mm}$ sobre el ámbito de estudio obtenido (las semillas, que son un subconjunto del dominio de m). El siguiente paso es la validación y experimentación, y éste es el momento en el que las hipótesis obtenidas mediante síntesis de las sentencias de mm, son validadas por el modelizador contrastándolas con una muestra de los originales de m y este proceso no es otra cosa que el constructor analítico $\mathcal{AC}_m$.

Como puede verse, ahora la relación entre $\mathcal{SC}_{mm}$ y $\mathcal{AC}_m$ y entre semillas y $\mathcal{O}_m$ es clara: el modelizador usa primero un método sintético ($\mathcal{SC}_{mm}$) para obtener las hipótesis a partir de un análisis y deducción de las sentencias de mm particularizándolas mediante una muestra de m, y a continuación realiza un proceso analítico para validar esas hipótesis ($\mathcal{AC}_m$).

- **Sintéticos/Completos**: cuando m es sintético (o completo), el $\mathcal{SC}_{mm}$ deberá generar también el proceso de creación $\mathcal{SC}_m$ de los originales de m, es decir, cada ejecución de $\mathcal{SC}_{mm}$ deberá crear tanto las sentencias como el constructor sintético de m, $\mathcal{SC}_m$, por lo que las sentencias de mm deben incluir también sentencias para la selección y ámbito de las semillas de $\mathcal{SC}_m$.

  En el caso de los modelos sintéticos se presume la existencia previa del modelo, pero no hemos indicado nada sobre cuál es su procedencia, al contrario

que los analíticos donde el modelo se construye tomando como referencia a los originales existentes previamente. Los metamodelos permiten dar una explicación a la procedencia de un modelo sintético, más allá del *pensamiento creativo* o del constructor sintético por prototipo: el constructor sintético de mm crea modelos sintéticos para que a su vez éstos puedan crear originales.

- **Analógicos**: la última posibilidad es que $\mathcal{SC}_{mm}$ cree modelos analógicos. Por las mismas razones que se indicaron para los modelos analíticos, en este caso el $\mathcal{SC}_{mm}$ debe estar relacionado con el $\mathcal{AgC}_m$ y sus modelos de referencia con las semillas utilizadas por $\mathcal{SC}_{mm}$. La explicación a estas relaciones son similares a las de los modelos analíticos e igualmente se basan en lo expuesto sobre el modelo científico. Efectivamente allí indicábamos que:

  > Si se parte de hipótesis ya contrastadas en otro ámbito de estudio pero se deduce que son aplicables de manera total o parcial al *ámbito de estudio* de la investigación, estaríamos nuevamente en un caso de analogía. (ver sección 5.2.1)

  De manera similar, si se parte de sentencias ya contrastadas del metamodelo pero se deduce que su $\mathcal{SC}_{mm}$ toma como semilla un modelo de referencia cuyos originales pertenecen a un dominio diferente al del modelo que sintetiza, entonces estaríamos en un caso de analogía.

Constructor analógico de metamodelo ($\mathcal{AgC}_{mm}$)

Cuando un metamodelo es analógico (en cualquiera de sus tres modalidades descritas en la sección 6.5.2, tendrá que existir un metamodelo de referencia que permita describir por analogía modelos existentes pero que no pueden ser observados (y por tanto analizados) por alguna razón (por ejemplo, por la imposibilidad de describirlos por parte de su modelizador o porque su complejidad es tan elevada que se recurre a modelos de referencia más simples).

7.2.2. Propiedades de metamodelo

En esta sección vamos a analizar qué propiedades tiene la relación de metamodelización.

1. **No es reflexiva**: efectivamente, en general no es cierto que $m \models m$ para cualquier metamodelo. Tampoco es *antireflexiva* puesto que nada impide

que existan *modelos reflexivos*, es decir que se modelan a sí mismos.

2. **Es intransitiva**: pero no *antitransitiva* como afirma Kühne (2006a), ya que si $mmm \models mm$ y $mm \models m$ no se puede afirmar que $mmm \models m$ sea cierto para cualquier m, mm, mmm. En el apartado 7.2.3 introduciremos el concepto de *modelos universales en un dominio* gracias al cual podría cumplirse la propiedad transitiva en algunos casos (ver propiedades de los modelos universales en el apartado 7.2.3).
3. **No es simétrica**: para cualquier modelo, si $mm \models m$ no implica que $m \models mm$
4. **Es antisimétrica**: puesto que $mm \models m$, $m \models mm$ sólo podría ser cierto si $mm = m$, es decir si el modelo fuese reflexivo.

7.2.3. Modelos universales. El modelo de sistema

En la sección 4.3 describimos el debate existente en la literatura acerca de los niveles de la modelización y la cuestión del modelo raíz. Efectivamente, se trata de un problema clásico en la literatura sobre modelización discutido desde hace más de una década (Bézivin and Lemesle, 1998). El punto de vista de OMG dividiendo las entidades en 4 niveles jerárquicos, donde el nivel más alto es recursivo, es una de las soluciones ofrecidas a esta cuestión. En ella el ámbito del problema, que podemos llamar *ámbito OMG*, se reduce a un subconjunto de $\mathcal{E}$ formado por entidades que son instancias o clases y en el que toda clase es a su vez instancia de una o más clases que se encuentran en el nivel inmediatamente superior. Para limitar el número de niveles, el nivel superior (MOF) contiene aquellas clases que son originales de sí mismas. Si nos fijamos en la figura 6.5 donde representábamos en general los universos de originales y modelos, veríamos que el *ámbito OMG* sería un subconjunto de $\mathcal{O}$ puesto que en ese ámbito toda entidad es un original (sin perder de vista, que desde nuestro punto de vista las clases del *ámbito OMG* habría que llamarlas protomodelos). Por otro lado, también vimos que en el *ámbito OMG* había mucha redundancia (replicación de conceptos), ya que la clase *class* aparecía en todos los niveles (excepto el inferior donde sólo hay instancias que no son modelos).

Nuestro punto de vista es algo más general al considerar que existen ámbitos donde no se producen estrictamente esas dependencias entre niveles. Consideremos el ejemplo de modelo de sistema. La gran mayoría de autores coinciden en la

definición de lo que es un sistema, siendo una de las más conocidas la propuesta por Klir (2001):

> a set of interacting units with relationships among them

Se trata de una definición muy general e intuitiva. Tanto es así que Klir (2001) llega a afirmar que el concepto de sistema forma parte de la esencia de lo que es el pensamiento humano. Efectivamente, es difícil imaginar alguna entidad que no pueda ser vista como un conjunto de elementos relacionados entre sí, excepto:

- las entidades atómicas (que no están formadas por otras entidades).
- los conjuntos simples en los que sus elementos no tienen definidas relaciones entre ellos.

En base a la anterior definición, también podríamos afirmar que el modelo de sistema es igualmente muy general. La mayoría de las personas podrían validar si una entidad es un original del modelo de sistema (reconocer elementos y relaciones) y también podrían intuitivamente analizar y sintetizar así como establecer analogías a partir de otros sistemas de referencia. Podemos decir, que el modelo de sistema es efectivamente muy general puesto que es un modelo cuyo *dominio* es muy amplio. Así mismo, también puede observarse que es recursivo, puesto que según nuestra teoría todos los modelos están formados por unos elementos (protomodelo, sentencias, características, constructores) relacionados entre sí.

El carácter general del modelo de sistema nos lleva a una cuestión interesante que aún no habíamos abordado. En nuestra definición de metamodelo indicábamos que éste es un modelo en el que *todos* sus originales son a su vez modelos. En contraposición, podríamos también distinguir un tipo de modelos en los que *todos* sus originales *no* son modelos, sino sólo originales. La cuestión es que si el modelo de sistema permitiese modelizar, tal como se intuye, a cualquier entidad incluido sí mismo, sea modelo o no, entonces no podemos decir estrictamente que sea un metamodelo. Obsérvese además que el dominio del modelo de sistema no es $\mathcal{E}$ puesto que al menos las entidades atómicas y los conjuntos simples no serían originales de sistema, tal como hemos indicado. Llamaremos a este tipo de modelos *modelos universales en un dominio*.

Definición **7.5.** Modelo universal en un dominio
Dado un conjunto de entidades $\mathcal{E}_d \subset \mathcal{E}$, diremos que un modelo $\alpha \in \mathcal{E}_d$ es universal en $\mathcal{E}_d$ si $\forall b \in \mathcal{E}_d, \alpha \models b$, siendo $\models$ la relación de modelización definida en el apartado 6.6.2

Veamos algunas de las propiedades de un modelo universal en un dominio:

- No tiene por qué ser un metamodelo, ya que en general no todos sus originales tienen que ser modelos. Por tanto, su función de validación debe indicar unas condiciones que puedan ser satisfechas tanto por modelos como por originales que no son modelos e igualmente sus constructores deben ser capaces de tratar modelos y originales no modelos. Sin embargo, pueden existir modelos universales en un dominio en el que todas las entidades sean modelos en cuyo caso también sería un metamodelo (metamodelo universal en un dominio).
- Un modelo universal es reflexivo, ya que de la definición anterior $\alpha \in \mathcal{E}_d$ se deduce que $\alpha \models \alpha$.
- No tiene por qué ser universal único en ese dominio. Sin embargo, todos los modelos de su clase de equivalencia serán universales en ese dominio, ya que $\forall m \in [\alpha], \mathcal{O}_m = \mathcal{E}_d$.
- Si forma parte de una terna de modelos, se produce una relación de modelización transitiva entre ellos. En efecto, si α es universal en $\mathcal{E}_d$ y $a, b \in \mathcal{E}_d$, el modelo $a \models b$, entonces es cierto que $\alpha \models a$, $a \models b$ y $\alpha \models b$. Para el caso de metamodelo universal en un dominio, podríamos afirmar que la relación de metamodelización es intransitiva en lugar de antitransitiva, tal como adelantamos en el apartado 7.2.2.
- Respecto al caso en el que el dominio sea $\mathcal{E}$, que podríamos llamar *Modelos universales absolutos*, nuestra conjetura es que existen entidades en $\mathcal{E}$ que no son ni originales ni modelos y por tanto sería contradictorio que existiese un modelo universal absoluto.

Volviendo al ejemplo de sistema, podemos afirmar que se trata de un modelo universal en un dominio que podemos llamar *Dominio sistémico* y que abarca a la mayoría de las entidades. El modelo de sistema tiene las propiedades de ser reflexivo y mínimo (relativamente simple) que son dos de las características debatidas en la literatura. La tercera característica a evaluar es la de si es un modelo raíz. Desde nuestro punto de vista, los modelos universales en un dominio no tienen, en general, por qué ser modelos raíz, puesto que este tipo de modelos sólo tiene sentido cuando se establece una agrupación de modelos y originales en niveles (como la arquitectura de niveles de modelización de 4 capas de OMG ya mencionada). Según la Definición 7.5, un modelo universal en un dominio en el

que además hayamos definido niveles, no sólo estaría en la *cima* de la jerarquía (de ahí el nombre alternativo de *top models*) sino también en cualquiera de los otros niveles incluyendo el nivel más inferior. Podríamos decir que también es un *bottom model*. La razón de esto es que no se cumpliría lo que algunos autores denominan *strict metamodeling* (Atkinson and Kühne, 2002b), es decir, que los modelos de una capa solo modelizan los modelos (u originales en la última) del nivel inmediatamente inferior.

Obsérvese que el aspecto clave de la definición de modelo universal es la delimitación de su universalidad en un dominio determinado. Esto implica que un determinado modelo puede ser universal en un dominio y no serlo en otros. Esa restricción es definida por el modelizador con el fin de simplificar el estudio de su problema mediante modelización. La identificación de modelos universales permite así mismo evitar la replicación de conceptos a la que hemos hecho referencia en el caso de OMG, puesto que no es necesario definir con diferentes nombres un mismo modelo.

7.3. Modelos característicos de un modelo

En el apartado 6.1.4 indicamos que las características de un protomodelo (y por tanto de un modelo) permiten al modelizador, evaluar la verdad o falsedad de las sentencias respecto a una determinada entidad. En este apartado estudiaremos una relación entre modelos que tiene que ver con las características de un modelo y que simplifica la labor del modelizador, especialmente en el proceso de análisis. Consideremos nuevamente el modelo de mesa. Una de sus características es *pata* que se refiere a los elementos que sujetan la superficie de la mesa. Por otro lado, en el modelo de *silla* también podemos distinguir una característica *pata* para referirnos a los elementos que sujetan la superficie de la silla donde alguien se sienta. Si creásemos los modelos de *mesa* y de *silla* de manera independiente nos encontraríamos que habría un conjunto de sentencias referidas a la característica *pata* que serían comunes a ambos modelos y también habría acciones en los constructores que serían las mismas para *mesa* y *silla*. El modelizador podría entonces definir un modelo de *pata* que le permita hacer uso de su propio protomodelo así como de sus propios constructores, independientemente de los de mesa o silla.
En ese caso, podríamos reescribir los modelos de mesa y silla haciendo uso, o *reutilizando*, el modelo de *pata*. El resultado es que las respectivas sentencias

referidas a *pata* en ambos modelos serían sustituidas por una única sentencia indicando que el valor de la *característica pata* en estos modelos cumple la función de validación del modelo de *pata*.
Llamaremos *modelos característicos* de un modelo a este tipo de modelos.

Definición **7.6.** Modelo característico de un modelo
Dado un modelo m, y c una característica de m, diremos que f_m es un modelo característico de m si y solo si existe una sentencia del protomodelo de m tal que el valor de la característica c en un original de m es un original de f_m. Así mismo, deberá cumplirse que los constructores de m deben hacer uso de los constructores de f_m. Denotaremos como $f_m \partial m$ la relación entre f_m y m.

Un modelo característico de un modelo tiene existencia por sí mismo y no difiere del resto de los modelos, por lo que tiene su propio protomodelo y sus propios constructores. Nótese que en la definición no estamos diciendo que el conjunto de sentencias del modelo característico sea un subconjunto de las sentencias del modelo caracterizado (en ese caso se trataría de una relación de submodelización), aunque es cierto que las sentencias del modelo característico tienen una presencia implícita en el modelo caracterizado.

Conviene resaltar que f_m no *modeliza* la característica c de m, sino que el modelizador de m *hace uso de un modelo de esa característica* o delega f_m para construir el protomodelo y los constructores de m.

Resulta lógico que también tenga que existir una relación entre los constructores de f_m y los de m, puesto que f_m está especializado en analizar, sintetizar o construir analógicamente respecto a esa característica. Del mismo modo que el protomodelo de m delega en el protomodelo de f_m, los constructores de m delegan en los constructores de f_m.

La caracterización de modelos es una forma de abstracción, puesto que el modelo se centra en las relaciones entre sus características en lugar de entrar en el *detalle* de las mismas. De manera recursiva, los modelos característicos pueden a su vez delegar en otros modelos característicos. Si el modelizador recogiese en el modelo de partida todas las sentencias y constructores de todas las características y las características de éstas recursivamente, obtendría un modelo confuso y no podría centrarse en su objetivo. Mediante los modelos característicos se obtienen modelos más simples puesto que el modelizador puede validar las características sin que tenga que conocer los detalles asociados a cada una de ellas.

Si las únicas sentencias de un modelo son las que hacen referencia a los modelos

característicos, estaríamos en un caso extremo en el que el modelizador se centra en las relaciones de las características que son relevantes para su modelo. Por ejemplo, en el modelo de mesa el modelizador evita incluir sentencias que se refieran a características de las patas en sí y se centraría en las relaciones entre las patas y los otros elementos de una mesa.

7.3.1. Propiedades de modelo característico de un modelo

Paradójicamente, la principal propiedad de los modelos característicos de un modelo es que no cumple las principales propiedades de las relaciones, tal como vemos a continuación:

1. **No es reflexiva**: efectivamente, en general no es cierto que $m \partial m$ para cualquier modelo. Sin embargo, nada impide que existan *modelos caracterizados por ellos mismos*, es decir, aquellos en los que alguna de sus características es modelizada por el mismo modelo.
2. **No es transitiva**: si $f_1 \partial f_2$ y $f_2 \partial m$ no se puede afirmar que $f_1 \partial m$ sea cierto para cualquier f_1, f_2, m, puesto que m ni siquiera tiene por qué tener la característica f_1.
3. **No es antisimétrica**: puesto que si $f_m \partial m$ y $m \partial f_m$ no implica necesariamente que $f_m = m$. Efectivamente, m podría hacer uso de f_m para una de sus características y f_m podría hacer uso de m para alguna de las suyas pero eso no implica que sean el mismo modelo.
4. **No es simétrica**: para cualquier modelo, puesto que si $f_m \partial m$ no implica que $m \partial f_m$

7.4. Transformación de modelos

Finalmente, haremos una breve mención a uno de los temas que ha despertado mayor interés en el ámbito de la modelización debido al papel relevante de MDA y MDE, tratados en el capítulo 2. Nos estamos refiriendo a la transformación de modelos. Aunque este tema podría ser tratado en mayor profundidad, nuestra intención en esta sección es únicamente describir la transformación desde el punto de vista de nuestra teoría considerando que es un caso particular de relación entre modelos.

El concepto de transformación es muy genérico: significa el cambio o alteración de una entidad (que puede estar formada a su vez por un conjunto de entidades) para dar como resultado otra entidad. Generalmente está asociado al concepto de función (Bottoni et al., 2006; Rensink, 2005). En efecto, según indica Bottoni et al. (2006), si denominamos $\mathcal{T}$ a una transformación definida mediante la función $T : \mathcal{E} \to \mathcal{E}$, tendremos que una entidad $s \in \mathcal{E}$ es el *origen* (*source*) y la entidad $t = T(s)$ sería el *destino* (*target*) de la transformación. El concepto es intuitivamente sencillo, sin embargo tal como indica Favre and Nguyen (2004):

> there is no consensual terminology and this term is often used to refer to distinct concepts

En nuestra opinión, efectivamente el concepto de transformación, en un sentido genérico puede referirse a diferentes casos en función de la existencia del *origen* y el *destino*, antes y después de la transformación. A continuación, proponemos una clasificación y una nomenclatura (en español e inglés) de las transformaciones según este criterio:

- **Correspondencia** (*mapping*): s y t existen antes y después de la transformación. En este caso, la transformación está asociada a un proceso previo para descubrir una regla que permita asociar ambas entidades o cómo se podría llegar a una a partir de la otra.
- **Sustitución** (*substitution*): s y t existen antes de la transformación pero s deja de existir tras la misma.
- **Derivación** (*derivation*): s existe antes de la transformación y t no. Tras la transformación s y t existen. En este caso diremos que t se ha creado a partir de s y por tanto es una derivación de s.
- **Transmutación** (*transmutation*): s existe antes de la transformación y t no. Después de la transformación s deja de existir, es decir s toma una nueva existencia desapareciendo su existencia anterior. Transmutaciones consecutivas en el tiempo diremos que dan lugar a una *evolución* (*evolution*).

Nótese que estas definiciones están muy relacionadas con la igualdad y la identidad de entidades: ¿cuál es el criterio para determinar si tras una transformación tenemos una nueva entidad o es la misma entidad que ha sido cambiada? Se trata de una cuestión filosófica aplicable a muchos ámbitos. Por ejemplo, cuando una persona crece ¿son el bebé de 2 meses y el anciano en el que se convierte con el paso de los años, la misma persona? ¿Cómo podemos saber que *es la misma*? La

respuesta podría ser: en realidad no son la misma persona (de hecho no se parecen en nada), pero él mismo y los demás consideran que sí lo es. Hay dos criterios clave para realizar o no esa diferenciación: 1) que exista una característica que permanece invariable tras la transformación (por ej. el ADN en el ejemplo) 2) que la propia entidad tenga *recuerdo o registro* de su evolución.

Veamos un ejemplo referido a los modelos. Una vez más consideremos el modelo de mesa. Definimos la transformación RP = *reducir una de sus patas* consistente en eliminar una de sus patas para convertirla en una mesa con una pata menos. Podemos aplicar esta transformación a un determinado modelo de mesa como por ejemplo el modelo de mesa de 4 patas, que describe a todas las mesas con ese número de patas y que llamaremos $\mathcal{M}_4$. El resultado sería un modelo de mesa con 3 patas, que llamaremos $\mathcal{M}_3$. Dicho modelo tendría diferentes sentencias del protomodelo y diferentes constructores y por tanto sería diferente a $\mathcal{M}_4$. La transformación de $\mathcal{M}_3$ a $\mathcal{M}_4$ podría ser de cualquiera de los tipos que hemos descrito anteriormente, si tenemos en cuenta que un modelo puede dejar de existir (trataremos con más detalle la existencia de modelos en la sección 9.6).

Es interesante analizar algunas de las propiedades de las transformaciones de modelos desde el punto de vista de nuestra teoría:

- Todas las relaciones entre modelos descritas en este capítulo y todas aquellas que podrían definirse pueden ser vistas como un caso especial de transformación de modelos. Así por ejemplo, podríamos definir una transformación que permitiera obtener un determinado supermodelo a partir de uno de sus submodelos.
- Tras el refinamiento de un modelo a través de su constructor analítico, determinar si el resultado son dos modelos o una *transmutación* del actual, depende del modelizador. Si decide que el modelo *origen* desaparezca será debido a que se cumplen las condiciones que hemos mencionado (invariancia de características y/o recuerdo o registro de su evolución).
- Las transformaciones, tal como han apuntado diversos autores (véase por ejemplo, Bézivin et al. (2006); Favre and Nguyen (2004); Mens and Van Gorp (2006)), también pueden ser modelizadas. Desde el punto de vista de nuestra teoría, se trataría de un metamodelo sintético cuyas semillas son los modelos *origen* y cuyos originales son los modelos *destino*. Obsérvese que el caso de *sustitución* de modelos puede realizarse como una *correspondencia* seguida de una destrucción del modelo *origen*. Del mismo modo,

una *transmutación* puede realizarse como una o *derivación* seguida de una destrucción del modelo *origen*.

Capítulo 8

Modelos, lenguajes y lenguajes de modelización

La exposición de una teoría de la modelización no es posible sin la utilización de un lenguaje. El presente texto utiliza esencialmente los lenguajes natural, matemático y gráfico para expresar el propio modelo de la modelización. A lo largo de los anteriores capítulos hemos sido conscientes de la dificultad por parte de un lector para imaginar que cualquier modelo, incluido el que se expone en esta memoria, existe sin que tal existencia dependa del uso o no de un lenguaje para expresarlo. Al mismo tiempo hemos aplicado el principio de independencia indicado en la sección 5.1 para demostrar que es posible una teoría de la modelización construida sin apoyarse en el concepto de lenguaje, ni siquiera para definir un metamodelo, a diferencia de una parte de la literatura que define metamodelo apoyándose en lenguaje de modelización. Esto no quiere decir que no exista una relación entre modelo y lenguaje en general o entre metamodelo y lenguaje de modelización. Lo que pretendemos mostrar es que mientras hemos definido modelo (y por tanto metamodelo) sin apoyarnos en el concepto de lenguaje, proponemos que, a la inversa, el concepto de lenguaje se apoye en el de modelo según los fundamentos de la modelización que hemos descrito hasta el momento.

Por tanto, el propósito de este capítulo es desarrollar ese enfoque profundizando en el concepto de lenguaje en relación con la teoría de la modelización expuesta.

El estudio del lenguaje es complejo y nos lleva una vez más a otras disciplinas como la filosofía o la semiótica a las que ya hemos hecho referencia. Aunque propondremos una definición de lenguaje desde la óptica de la modelización, nuestro

foco no estará tanto en redefinir lenguaje como en ofrecer un nuevo punto de vista que permita comprender mejor el papel que juegan lenguaje y modelo en el proceso de la comunicación. Los conceptos descritos serán además el punto de partida para desarrollar en el capítulo 9 el tema de la *pragmática de la modelización*.

8.1. Expresión de modelos

Desde un punto de vista informal un lenguaje es una herramienta de comunicación que permite a los modelizadores expresar sus modelos para transmitirlos a otros sujetos (humanos o máquinas) que puedan interpretarlos. Llamaremos *modelos expresados* (*expressed models*) a este tipo de modelos.

No todos los modelos son *expresados* puesto que los modelizadores pueden crear y usar modelos que no necesitan, no quieren o no pueden ser comunicados. A este tipo de modelos, los denominaremos *modelos inexpresados* (*unexpressed models*). Para ilustrar la existencia de los modelos inexpresados veamos algunos

Figura 8.1: Reconocimiento facial de hombre o mujer o viceversa

ejemplos. Consideremos en primer lugar el modelo de una cara para ser usado en reconocimiento facial. Todos los humanos somos capaces de reconocer caras incluso en circunstancias muy adversas tales como cuando distinguimos a una persona que no hemos visto en años en una multitud. No sólo somos capaces de diferenciarla del resto de caras en un grupo sino también realizar el reconocimiento a pesar de los cambios que se han producido en su rostro con el paso de la edad. Del mismo modo, somos capaces de diferenciar determinados tipos de caras como por ejemplo si es de hombre o mujer, de una determinada raza o incluso si es de una determinada familia. La figura 8.1 ilustra el reto de distinguir entre la cara de

hombre y mujer, y que la mayoría de los humanos puede superar sin que muchos de ellos puedan explicar cuáles son las razones de su elección.

Tras esa capacidad subyace el modelo de reconocimiento de la cara de una persona concreta que nosotros (modelizadores) creamos en nuestra mente. Se trata de modelos complejos que todo el mundo es capaz de crear y usar desde muy temprana edad. Podemos distinguir los constructores de estos modelos:

- $\mathcal{AC}_m$: obtiene el protomodelo de reconocimiento de la cara de una persona
- $\mathcal{V}_m$: permite al modelizador determinar si una determinada cara corresponde con la de la persona a la que está referido el modelo
- $\mathcal{SC}_m$: aunque el modelo de reconocimiento facial es típicamente analítico, también pueden existir modelos sintéticos, como por ejemplo el que usa un dibujante para obtener las caras de un personaje de su invención. Incluso podría ser un modelo completo si por ejemplo un pintor realiza diferentes cuadros de una misma persona basándose en el modelo de su cara. En este caso, tanto el original percibido (que es la percepción de la cara real por el pintor) como los originales concebidos (las pinturas realizadas) son originales del mismo modelo. Podríamos decir que podemos percibir diferentes *caras* (originales) de la misma persona según la veamos directamente o por la pintura realizada a partir de ésta por el pintor (modelizador).
- $\mathcal{AgC}_m$: por ejemplo, a partir de una cara existente un modelizador (por ejemplo, nuevamente un dibujante) inventa una cara nueva inspirada en el modelo de aquella (modelo de referencia).

Lo interesante de estos modelos es que a pesar de que son creados y usados a diario por cualquier persona, no somos capaces de expresarlos completa ni apropiadamente mediante un lenguaje. Si se hubiesen conseguido expresar completamente estos modelos, habría sido posible desarrollar sistemas informáticos infalibles para el reconocimiento facial automatizado. Actualmente estos modelos fallan mientras que bebés recien nacidos pueden reconocer sin error a su madre aunque no puedan explicar cómo lo hacen. Según lo explicado en el apartado 7.1.3 diremos que los modelos automáticos y los realizados por una persona no son equivalentes. Aunque podemos transmitir algunas de las sentencias del modelo a otra persona o a una máquina (por ej. es morena, de tez pálida, nariz pequeña, ojos azules) eso no es suficiente para que un tercero siempre pueda reconocerla con éxito entre un grupo de personas (que podrían satisfacer esas sentencias) pues siempre hay

alguna sentencia relativa a alguna característica difícil o imposible de expresar y que la distinguiría de todas las demás. Según lo descrito en la sección 7.1 todos esos modelos *fallidos* en realidad son supermodelos del verdadero modelo que estamos buscando.

Otro ejemplo interesante de modelo inexpresado es el modelo de la realidad que toda persona construye a lo largo de su vida. Se trata de un modelo muy personal (en el sentido de privado o íntimo) y muy complejo que rara vez se comunica a otras personas, porque no se quiere o porque es muy difícil de expresar. Estos modelos son muy numerosos puesto que hay tantos modelos diferentes como personas han existido, existen y existirán. Sin duda sería muy interesante expresar esta clase de modelos para ser capaces de aprender de ellos, especialmente de los modelos de aquellas personas que vieron el mundo de una manera extraordinaria como Einstein o Leonardo Da Vinci. Por desgracia, esos modelos *murieron* con su modelizador y por tanto no será posible expresarlos en un lenguaje y transmitirlos al resto de la humanidad. Una vez mostrado que un modelo puede existir sin ser expresado, cabe preguntarse por qué los términos modelo y lenguaje aparecen juntos en buena parte de la literatura sobre modelización. La respuesta a esta pregunta es compleja y por tanto nos iremos aproximando a su solución desde nuestro punto de vista, de una manera progresiva mediante la introducción de algunos conceptos básicos.

8.2. El metamodelo lingüístico: modelos lingüísticos

Una forma habitual de abordar el problema de los lenguajes consiste en describir sus aspectos lingüísticos, los cuales son generalmente articulados mediante la formulación de un conjunto de reglas cuya descripción se conoce habitualmente como *gramática* (Harel and Rumpe, 2000, 2004). Por esta razón, entendemos que en nuestra teoría de la modelización para abordar la definición del término *lenguaje*, o al menos para establecer lo que será la base para tal definición, ha de proponerse un modelo que refleje la estructura de reglas de un modo genérico. Este modelo es realmente un metamodelo que denominaremos como *el metamodelo lingüístico*. En los siguientes apartados definiremos dicho metamodelo y describiremos en detalle sus constructores.

8.2.1. Definición de Metamodelo Lingüístico. Protomodelo

Definición **8.1.** Metamodelo lingüístico
Se define el Metamodelo lingüístico $\mathcal{LMM}$ (*Linguistic MetaModel*) como un modelo cuya función de validación $\mathcal{V}_{\mathcal{LMM}}$ está determinada por las siguientes sentencias referidas a los originales de $\mathcal{LMM}$ o Modelos lingüísticos, que denotaremos $\mathcal{LM}$ (*Linguistic Model*):

- $d_1 = \mathcal{LM}$ tiene un conjunto denominado **alfabeto** formado por un conjunto finito de **símbolos**.
- $d_2 = \mathcal{LM}$ tiene un conjunto de **reglas léxicas** que establecen cómo se combinan los símbolos del alfabeto para producir otro símbolos denominados **símbolos léxicos** o **lexemas**. El conjunto de lexemas se denomina **vocabulario**.
- $d_3 = \mathcal{LM}$ tiene un conjunto de **reglas sintácticas** o **sintaxis** que definen cómo se combinan los lexemas para producir otros símbolos que serán los originales de $\mathcal{LM}$ y que denominaremos **expresiones**.
- $d_4 =$ si $\mathcal{LM}$ tiene un constructor analítico/analógico éste define cómo analizar expresiones basándose en una particularización de las reglas indicadas en las sentencias anteriores para crear un $\mathcal{LM}$ concreto
- $d_5 =$ si alternativamente $\mathcal{LM}$ tiene constructores sintéticos éstos permiten generar sus **expresiones** en base a las **reglas léxicas** y **reglas sintácticas** indicadas en las sentencias anteriores.

Obsérvese que al tratarse de un metamodelo, estas sentencias se refieren a la existencia de un conjunto de reglas no a cómo son esas reglas en un caso particular. Por lo tanto, un original del metamodelo lingüístico es un modelo, al que hemos denominado **modelo lingüístico** $\mathcal{LM}$, cuyas sentencias pueden ser formuladas por un sistema de reglas en dos niveles (léxico y sintáctico) a partir de un conjunto de símbolos o alfabeto y dichas sentencias conforman su protomodelo. Por otra parte, y conforme a lo descrito sobre los metamodelos en la sección 7.2.1, es necesario que para que $\mathcal{LMM}$ sea un metamodelo, sus sentencias indiquen no sólo las características de sus originales ($\mathcal{LM}$) sino también las de los constructores de estos modelos lingüísticos. Este hecho queda reflejado en las sentencias d_4 y d_5. En efecto, las sentencias de un cierto modelo lingüístico $\mathcal{LM}$ se obtienen particularizando las sentencias de $\mathcal{LMM}$, lo que dará lugar a un sistema

concreto de reglas de producción en dos niveles (léxico y sintáctico) basado en un alfabeto inicial concreto. El $\mathcal{SC}_{\mathcal{LM}}$ se estructura alrededor de ese sistema de producción y puede generar nuevas expresiones por la aplicación sucesiva de esos dos niveles de reglas. Para el caso de los modelos lingüísticos analíticos, su $\mathcal{AC}_{\mathcal{LM}}$, en virtud de lo descrito en la sección 7.2.1, estará basado en la ejecución del constructor sintético del metamodelo lingüístico $\mathcal{SC}_{\mathcal{LMM}}$, tal como veremos en el siguiente aparatado.

Podemos aplicar nuestra clasificación de modelos a los modelos lingüísticos. Por ejemplo, los lenguajes naturales se caracterizan por tener modelos lingüísticos de cualquier tipo (analíticos, analógicos, sintéticos o completos), teniendo estos modelos un alfabeto con un número relativamente pequeño de símbolos pero con una gran cantidad de reglas correspondientes a los dos niveles (léxico y sintáctico). Estas reglas generalmente cambian a lo largo del tiempo (por ejemplo, son enriquecidas con nuevos lexemas). Eso implica, como hemos indicado en el capítulo 6 que su $\mathcal{AC}_{\mathcal{LM}}$ está continuamente realizando un refinamiento del $\mathcal{LM}$. De hecho, el modelo lingüístico de un lenguaje natural nunca deja de actuar mientras el lenguaje esté *vivo* (en uso).

Como curiosidad, podemos también considerar los modelos lingüísticos $\mathcal{LM}$ de lenguajes no humanos como el de las abejas, mediante el que se comunican para indicar la localización de las flores a través de movimientos de abdomen. También es interesante el caso del Esperanto, cuyo modelo lingüístico $\mathcal{LM}$ es sólo sintético. Por otra parte, los lenguajes no naturales, tales como los de programación o XML, inventados para comunicar instrucciones o datos a los sistemas software son otros ejemplos de lenguajes puramente sintéticos.

$\mathcal{LMM}$ no impone ninguna restricción sobre la naturaleza de los símbolos. De hecho, los símbolos pueden ser de cualquier tipo, típicamente texto (Hopcroft et al., 2006), pero también gráficos, gestuales, etc. En el caso de los modelos lingüísticos relativos a lenguajes naturales las reglas a veces son complejas. En general, podemos afirmar de manera simplificada que el alfabeto está formado por letras, dígitos, etc., el vocabulario por palabras y el nivel sintáctico por reglas para la generación de frases (por ejemplo la regla de correspondencia entre sujeto y verbo, la de condicional subjuntivo, etc.). Para el caso de los modelos lingüísticos de lenguajes de programación, el alfabeto también son letras, dígitos, palabras reservadas, etc.; el vocabulario son los identificadores, valores, etc.; la sintaxis son las reglas para la composición de las estructuras de control, declaraciones,

expresiones, etc. Si el modelo lingüístico se corresponde con un lenguaje gráfico (como puede ser UML), el alfabeto consiste en símbolos gráficos simples (líneas abiertas o cerradas por ej.); el nivel léxico son los símbolos derivados del alfabeto (flechas, elipses, cajas, etc.); las reglas sintácticas nos permiten combinar estos símbolos léxicos para construir diagramas (Harel and Rumpe, 2000)

8.2.2. Definición de Metamodelo Lingüístico. Constructores.

Los constructores del modelo lingüístico a los que hacen referencia las sentencias d_4 y d_5 no deben confundirse con los constructores del propio metamodelo lingüístico aunque como vimos en la sección 7.2.1 existan relaciones con ellos.

$\mathcal{LMM}$ es un modelo analógico completo cuyos constructores son los siguientes:

- **Constructor analógico** $\mathcal{AgC}_{\mathcal{LMM}}$: el constructor analógico de $\mathcal{LMM}$ define cómo se obtienen las propias sentencias indicadas en la Definición 8.1. La estructura de reglas, común a todos los modelos lingüísticos ha sido objeto de estudio durante siglos y aunque puede expresarse de diferentes formas podemos afirmar sin duda que esta estructura genérica constituye el metamodelo lingüístico. En nuestro caso para la formulación de esas sentencias nos hemos basado principalmente en el trabajo de Harel and Rumpe (2000) (que ha actuado como modelo de referencia de nuestro metamodelo analógico) donde se mencionan 4 niveles de reglas cuyas dos últimas hemos resumido en 2 debido a sus similitudes. Por tanto, los detalles sobre el constructor analítico del modelo de referencia del metamodelo lingüístico pueden encontrarse en la abundante literatura sobre lingüística de la que (Harel and Rumpe, 2000) es sólo un ejemplo. Muchos de estos autores han sido los modelizadores de este modelo de referencia. A modo de resumen podríamos decir que el constructor analítico del modelo de referencia es un proceso complejo en dos pasos donde en primer lugar se analizan un conjunto de expresiones de la misma naturaleza hasta comprobar que todas ellas fueron generadas a partir de un conjunto de símbolos y en segundo lugar siguiendo un conjunto de reglas de producción determinadas estos símbolos eran agrupados en lexemas, luego en símbolos sintácticos y finalmente en expresiones, siendo el resultado de todo ello un determinado modelo lin-

güístico. Aplicando este proceso sobre diferentes muestras de expresiones se puede llegar a la conclusión de que todos los modelos lingüísticos comparten la misma estructura en niveles que hemos expresado en la Definición 8.1.

- **Constructor sintético $\mathcal{SC}_{\mathcal{LMM}}$**: el metamodelo lingüístico también incluye un constructor sintético mediante el cual puedan crearse nuevos modelos lingüísticos. Este constructor tomaría como entrada un conjunto de semillas para obtener los modelos lingüísticos (protomodelo y constructores). Podemos considerar dos casos diferentes de constructores sintéticos. En primer lugar el modelizador de $\mathcal{LMM}$ puede tomar unas semillas consistentes en un conjunto de símbolos (que constituirán el alfabeto) y un conjunto de reglas que pasarán a ser sucesivamente las reglas léxicas y sintácticas del nuevo modelo lingüístico. Esta elección la haría tras un proceso creativo. El segundo caso interesante es cuando el modelizador toma un conjunto lo suficientemente relevante (muestra) de expresiones en un ámbito determinado, las analiza y a partir de ahí ejecuta el constructor sintético de $\mathcal{LMM}$ que genere el alfabeto y las reglas léxicas y sintácticas del modelo lingüístico $\mathcal{LM}$ obtenido. Obsérvese que se trata de un proceso analítico que dará lugar al constructor analítico del modelo lingüístico $\mathcal{LM}$ y que refleja las relaciones existentes entre $\mathcal{SC}_{\mathcal{LMM}}$ y $\mathcal{AC}_{\mathcal{LM}}$, así como la relación entre las muestras de $\mathcal{AC}_{\mathcal{LM}}$ (expresiones) y las semillas de $\mathcal{SC}_{\mathcal{LMM}}$, tal como explicamos en el apartado 7.2.1. Por ejemplo, este debió ser el caso del constructor sintético que creó Jean-François Champollion para obtener el alfabeto y las reglas léxicas y sintácticas de los jeroglíficos egipcios, tras el proceso de análisis de las expresiones contenidas en la piedra de Rosetta.

8.2.3. Submodelos lingüísticos

Al ser $\mathcal{LM}$ un modelo también puede tener submodelos, que llamaremos lógicamente **submodelos lingüísticos**.

La especificación de XML es un ejemplo de un modelo lingüístico: un documento XML concreto bien-formado es una expresión (original) del modelo lingüístico de XML ($\mathcal{LM}_{XML}$). Resulta interesante reflexionar sobre las particularizaciones de XML. Una forma habitual de restringir el conjunto de todas las expresiones válidas de una determinada particularización de XML es mediante el uso de un DTD o XML Schema. Cada particularización de XML, como por ejemplo MathML, es también un modelo lingüístico, con diversas reglas que especifican los nombres

permitidos de etiquetas, secuencia de elementos, cardinalidades, etc. Es obvio que todos los originales (expresiones) del modelo lingüístico de MathML ($\mathcal{LM}_{MathML}$) son también originales de ($\mathcal{LM}_{XML}$), puesto que también son expresiones XML. Por tanto, conforme a la definición de submodelo (Definición 7.1), podemos afirmar que $\mathcal{LM}_{MathML}$ es un submodelo de $\mathcal{LM}_{XML}$. Obsérvese que este ejemplo anticipa que nuestra teoría difiere de la opinión general, que clasifica XML como un metalenguaje (este término será definido más adelante en el apartado 8.4.4).

8.2.4. Características lingüísticas

En la sección 6.1.4 describimos el concepto de características en relación con los (proto)modelos como aquellas propiedades de los originales a las que hacen referencia las sentencias de un modelo. Para el caso concreto de los modelos lingüísticos podemos identificar esas características que lógicamente estarán referidas a las expresiones (originales) de esos modelos. Así en el $\mathcal{LM}$ de un lenguaje natural, las sentencias (que se corresponderán con reglas de alguno de los niveles), serán por ejemplo del tipo: *tiene un «sujeto» que aparece antes que el «predicado»*. Para el caso del $\mathcal{LM}$ de un lenguaje matemático serían por ejemplo: *tiene dos «operandos» que siempre deben estar relacionados por un «operador»*. O en el caso de los diagramas de caso de uso de UML: *todo «actor» deberá estar relacionado con al menos un «caso de uso»*. En estos ejemplos, los términos *«sujeto»*, *«predicado»*, *«operando»*, *«operador»*, *«actor»*, *«caso de uso»* son precisamente las características de esos modelos lingüísticos a las que hacen referencia sus sentencias.

Hemos usado los símbolos « y » para resaltar que las expresiones entre ellos encerradas son las características del modelo (y por tanto están en el plano conceptual) y como veremos en las secciones siguientes, no deben confundirse con los términos utilizados para expresarlas. Estas características constituyen, junto a las relaciones entre ellas, lo que algunos autores han denominado *sintaxis abstracta* (ver por ejemplo (Baar, 2006; Feilkas, 2006; Gargantini et al., 2009)), pero que nosotros llamaremos simplemente *características lingüísticas* y no son otra cosa que un caso particular de las caracteristicas de un modelo para el caso de los modelos lingüísticos.

8.3. Correspondencia semántica

Hemos visto que los originales de un modelo lingüístico son las expresiones. Todas las expresiones, en virtud de la estructura de reglas que impone el metamodelo lingüístico sobre los modelos lingüísticos, podrán ser analizadas y/o sintetizadas desde un punto de vista léxico y sintáctico. Sin embargo, las expresiones serán de total inutilidad si no se refieren a otras entidades, tal y como indica el triángulo del significado cuando establece un vínculo entre símbolo y cosa pasando por concepto (ver sección 5.1).

El principal objetivo de las expresiones es transmitir información, pero ¿quién o qué es el receptor de esa información? En el contexto de la comunicación, Overbeek (2006) denomina al modelizador como el *remitente* y al receptor como el *intérprete* (el modelizador es el creador del modelo y el intérprete el usuario de ese modelo que se transmite). La cuestión es que para que exista una verdadera comunicación tanto el remitente como el destinatario deben comprender el lenguaje utilizado. Pero, ¿qué significa *comprender* el lenguaje? Este problema se refiere a lo que se conoce como *semántica* del lenguaje. Se trata de un aspecto del lenguaje que se refiere a la asociación de símbolos con conceptos. A continuación abordaremos esta cuestión desde el punto de vista de nuestro enfoque de la modelización, introduciendo el concepto de *correspondencia semántica*.

8.3.1. Definición

Definición **8.2.** Correspondencia Semántica
Sea $\mathcal{O}_{\mathcal{LM}}$ el conjunto de expresiones (originales) de un modelo lingüístico $\mathcal{LM}$, y sea $\mathcal{SM}$ un conjunto formado por modelos que denominaremos **modelos semánticos** (*semantic models*). Se define **Correspondencia semántica** de $\mathcal{LM}$ ($\mathcal{S}em\mathcal{C}_{\mathcal{LM}}$) como un conjunto de relaciones denominadas **relaciones semánticas**, donde cada una de ellas asocia una expresión $ex \in \mathcal{O}_{\mathcal{LM}}$ con un modelo $sm \in \mathcal{SM}$. Cada relación $(ex, sm) \in \mathcal{S}em\mathcal{C}_{\mathcal{LM}}$ es un **significado** de ex y también un **significante** de sm. Denominaremos **expresiones con significado** (meaningful expressions) a las expresiones de $\mathcal{LM}$ que forman parte de alguna relación semántica de $\mathcal{S}em\mathcal{C}_{\mathcal{LM}}$ y **expresiones sin significado** (meaningless expressions) al resto.

Uno de los aspectos más relevantes de la definición anterior es que cualquier relación semántica debe conducir a un modelo, aunque sea tan específico como un

modelo unitario. Este enfoque difiere, aunque no es incompatible, con el adoptado por otros autores (ver por ej. (Gleitman and Papafragou, 2005; Harel and Rumpe, 2000)) quienes utilizan el término *concepto* como destino de la correspondencia.

La justificación de esta elección es consistente con lo descrito en la sección 5.1 sobre los triángulos del significado y de la modelización. En efecto, estos autores toman como referencia el triángulo del significado donde se usa concepto, símbolo y referente (o cosa). Nuestro enfoque se acerca al adoptado por Gupta and Sykes (2001) aunque con un matiz importante: mientras que Gupta and Sykes identifican modelo como un sistema de conceptos, nuestra definición de modelo incide en su aspecto pragmático, que será ampliado en el capítulo 9. En nuestra opinión, la mente humana no crea conceptos *porque sí* sino con la finalidad de obtener representaciones simplificadas de la realidad a través de un proceso de análisis, que luego puedan ser validadas frente a entidades reales y que también permitan, mediante un proceso de síntesis, obtener nuevas representaciones que posteriormente puede ser materializadas. Cuando se utiliza una expresión, que en virtud de $\mathcal{S}em\mathcal{C}_{\mathcal{LM}}$ tiene una correspondencia con un modelo, se comunica efectivamente un concepto, pero ese concepto tiene que llegar a ser también un modelo en la persona receptora de la comunicación, para que le pueda ser útil. En definitiva, nuestra propuesta es que las expresiones lingüísticas tienen que estar asociadas a modelos porque son el único tipo de conceptos que permiten al receptor hacer uso de ellos. Por tanto, se trata de un enfoque que no es incompatible con el triángulo del significado al ser todo modelo un concepto en sí mismo pero que pone el énfasis en su definición como *habilidad cognitiva* (Zalta et al., 2011) al que hicimos referencia en el apartado 6.1.2.

8.3.2. Nuestra propuesta de Triángulo de la modelización

Llegados a este punto podemos mostrar nuestra versión del triángulo de la modelización que introdujimos en la sección 5.1:

Como puede observarse en la figura, el proceso por el cual los originales pasan a ser modelos coincide con el proceso analítico mientras que el sentido inverso es el proceso sintético. Por otro lado, un modelo puede ser expresado (o no, de ahí la línea punteada y el color gris del vértice *expresión*) mediante la correspondencia semántica de un determinado lenguaje.

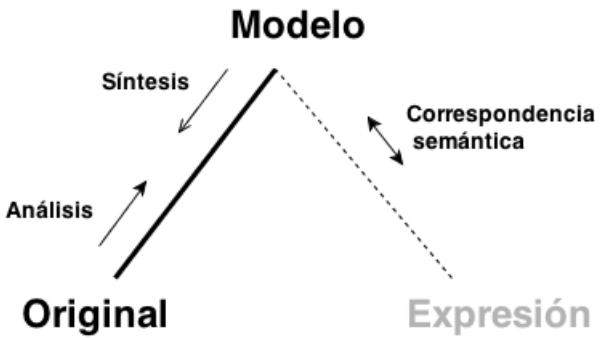


Figura 8.2: Nuestra propuesta de Triángulo de la modelización

8.3.3. Tipos de correspondencia semántica

En general, como es bien conocido, en una correspondencia donde denominamos origen al primer conjunto y destino al segundo, podemos preguntarnos por las siguientes propiedades:

1. **Unicidad de origen**: los elementos de destino están relacionados con un único elemento de origen
2. **Unicidad de destino**: los elementos de origen están relacionados con un único elemento de destino
3. **Existencia de origen**: todo elemento de destino está relacionado con al menos un elemento de origen
4. **Existencia de destino**: todo elemento de origen está relacionado con al menos un elemento de destino

Considerando estas propiedades para el caso de $\mathcal{S}em\mathcal{C}_{\mathcal{LM}}$ podemos decir que en general no se cumple ninguna de ellas dando lugar, respectivamente, a diferentes tipos de correspondencia semántica (que son precisamente la negación de las anteriores propiedades):

1. **Sinonimia**: un modelo semántico puede estar relacionado con más de una expresión
2. **Polisemia**: una expresión puede estar relacionada con más de un modelo semántico
3. **Modelos inexpresados**: algún modelo semántico no está relacionado con al menos una expresión
4. **Expresiones sin significado**: alguna expresión no está relacionada con al menos un modelo semántico

8.3.4. Homomorfismo entre características semánticas y lingüísticas

Es interesante resaltar que dada una correspondencia semántica $\mathcal{S}em\mathcal{C}_{\mathcal{LM}}$ podría existir otra correspondencia, esta vez entre las características de los modelos asociados a una determinada relación semántica. Supongamos que dada una determinada relación $(ex, sm) \in \mathcal{S}em\mathcal{C}_{\mathcal{LM}}$, nos fijamos en las características del modelo sm y en las características del modelo lingüístico $\mathcal{LM}$ de ex y observamos si podría existir una relación entre ambas. Se trataría de la posibilidad de una relación entre características lingüísticas y semánticas que ha sido indicada por algunos autores (véase por ejemplo (Chierchia and Turner, 1988)). Esta correspondencia podría ser además un homomorfismo si existiese no sólo una correspondencia entre las características sino si además se mantienen las relaciones entre éstas. Es decir, cuando de una relación entre características de un modelo semántico, se deriva igualmente una relación entre las características lingüísticas correspondientes. La existencia de este homomorfismo es un indicativo de la idoneidad de un modelo lingüístico para que sus expresiones puedan expresar, valga la redundancia, sus modelo semánticos correspondientes. Esta idoneidad será óptima cuando sea un isomorfismo, es decir cuando todas las características lingüísticas tengan su correspondencia en las características del modelo semántico y viceversa.

Así por ejemplo, para expresar convenientemente el modelo de una *lista ordenada* de elementos, necesitaríamos un modelo lingüístico que pudiese generar *listas (ordenadas)* de símbolos (separados típicamente por otros símbolos denominados separadores). El homomorfismo consiste en este caso en que la relación de orden existente entre los elementos de la lista del modelo semántico se mantiene entre la lista de símbolos de la expresión correspondiente.

8.4. Lenguaje

8.4.1. Definicion

Una vez definido modelo lingüístico y correspondencia semántica podemos definir Lenguaje de una manera directa.

Definición **8.3.** Lenguaje
Se define **Lenguaje** $\mathcal{L}$ como un par $(\mathcal{LM}_{\mathcal{L}}, \mathcal{S}em\mathcal{C}_{\mathcal{L}})$ compuesto por un modelo

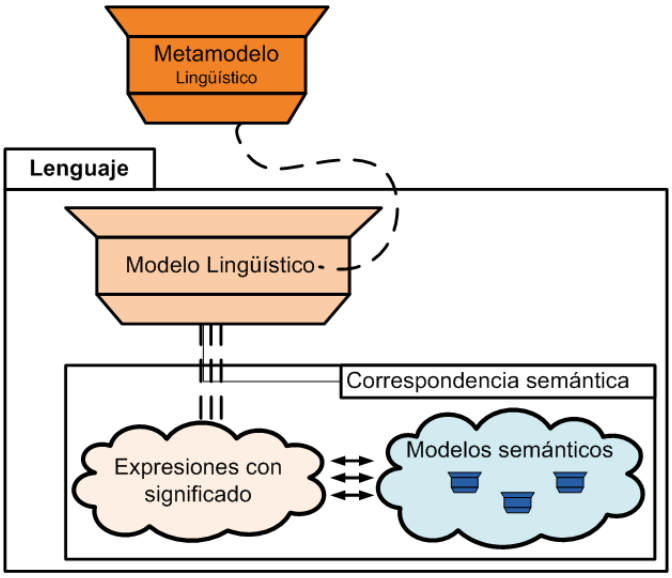


Figura 8.3: Representación gráfica de Lenguaje

lingüístico y una correspondencia semántica asociada. Se denominan **expresiones** del lenguaje a aquellas expresiones con significado de $\mathcal{LM}_{\mathcal{L}}$.

A diferencia de lo que puede observarse a partir del RCM de Lenguaje (ver apartado 4.1.3), para nosotros un lenguaje no es simplemente un sistema o conjunto de símbolos o frases (Chomsky, 1957; De Saussure, 1974; Favre, 2005; Harel and Rumpe, 2000; Hopcroft et al., 2006; Luria, 1979; Lyons, 2002; Nowak et al., 2002; Robins, 1971), sino más bien una entidad abstracta (Kühne, 2006a). En la figura 8.3 pueden observarse los elementos de esa entidad abstracta, que tiene asociado un modelo (su modelo lingüístico) a partir del cual pueden obtenerse o analizarse expresiones (que sí son un sistema de símbolos) y, sobre todo, tienen asociada una correspondencia semántica que es la que hace posible que puedan comunicarse modelos entre dos sujetos. Un lenguaje, por tanto, es la esencia de la comunicación. Sin lenguaje no puede existir comunicación, y lo que se comunican son un tipo especial de conceptos, los modelos, que permiten al receptor de la comunicación usarlos tal como los creó el emisor (modelizador) o de una manera diferente (analítica, analógica y/o sintética) y posiblemente también con otra finalidad.

Otros autores (Favre, 2005; Feilkas, 2006; Nowak et al., 2002) afirman que una gramática es el modelo de un lenguaje, definido mediante la especificación de un conjunto de reglas. Desde nuestro enfoque ese modelo podría asimilarse a lo que hemos definido como modelo lingüístico del lenguaje. Sin embargo, un

$\mathcal{LM}$ por sí solo no es un lenguaje puesto que las expresiones sin significado no podrían ser consideradas expresiones de un verdadero lenguaje. Así por ejemplo una expresión como *el universo fue al trabajo con ballenas de melón* es un original del modelo lingüístico del español, pero al tratarse de una frase a la que es difícil encontrar significado, no puede considerarse una expresión del lenguaje español.

8.4.2. Sublenguaje

Igualmente, a partir de las definiciones de lenguaje (Definición 8.3) y submodelo lingüístico (apartado 8.2.3) podemos definir sublenguaje como:

Definición **8.4.** Sublenguaje
Sean los lenguajes $\mathcal{L}$ y $\mathcal{L}'$, se dice que $\mathcal{L}$ es un **sublenguaje** de $\mathcal{L}'$ si y solo si $\mathcal{LM}_{\mathcal{L}}$ es un submodelo lingüístico de $\mathcal{LM}'_{\mathcal{L}}$ y $\mathcal{S}em\mathcal{C}_{\mathcal{L}} \subseteq \mathcal{S}em\mathcal{C}_{\mathcal{L}'}$

Ya hemos mencionado un ejemplo de submodelo lingüístico (MathML) en el apartado 8.2.3 y vimos que se produce una relación de submodelización entre los modelos lingüísticos de XML y MathML. Por otra parte el conjunto de los modelos semánticos de MathML $\mathcal{SM}_{MathML}$ está formado por los modelos de las construcciones matemáticas, mientras que $\mathcal{SM}_{XML}$ es un conjunto más amplio que contiene todos los modelos semánticos que pueden expresarse en XML y que por tanto incluye también a $\mathcal{SM}_{MathML}$. Por tanto, puede concluirse que el conjunto de las relaciones semánticas de MathML está incluido en el conjunto de las relaciones semánticas de XML en general, por lo que podemos afirmar que MathML es un sublenguaje de XML.

8.4.3. Lenguaje de modelización

Un término muy usado, que hemos mencionado en diversas ocasiones en esta memoria es el de lenguaje de modelización. La intuición sugiere que un lenguaje de modelización sirve para expresar modelos. Por tanto, y dado que en la correspondencia semántica de cualquier lenguaje se emparejan expresiones del lenguaje con modelos, podría deducirse erróneamente que cualquier lenguaje es de modelización. Sin embargo, sabemos que esto no es cierto puesto que hay muchos lenguajes, por ejemplo el español o el morse, que no son lenguajes de modelización.

¿Cuál es entonces la diferencia entre los lenguajes de modelización y el resto de los lenguajes?

La literatura ha tratado extensamente el concepto de lenguaje de modelización especialmente respecto a sus relaciones con el concepto de metamodelo. Efectivamente, tal como indicamos en el apartado 4.1.2 del RCM de Metamodelo podemos observar que hay dos grupos principales de definiciones. Por un lado, un grupo de autores (Aß mann et al., 2006; Gonzalez-Perez and Henderson-Sellers, 2007; Kühne, 2006a; OMG, 2003) identifican metamodelo con un modelo de modelos y otro grupo (Aß mann et al., 2006; Favre, 2005; Kurtev, 2007; Mellor et al., 2004; Ober and Prinz, 2006; OMG, 2002; Seidewitz, 2003) define metamodelo relacionándolo con lenguaje de modelización. Así por ejemplo, Seidewitz (2003) define metamodelo como un modelo de un lenguaje de modelización. Aunque, como veremos a continuación, de nuestra teoría se deduce también una relación entre metamodelo y lenguaje de modelización, la anterior definición no se corresponde con nuestra definición de lenguaje principalmente por dos razones:

1. Basándonos, una vez más, en el principio de independencia, los modelos son independientes de los lenguajes y por tanto, desde nuestro enfoque modelo no puede definirse a partir de lenguaje al tratarse aquél de un concepto más general. De hecho, como ya se ha indicado, un metamodelo puede existir sin un lenguaje de modelización.
2. Nuestra definición de metamodelo no difiere esencialmente de la de modelo, sino que es un caso particular en el que los originales son a su vez modelos y sus sentencias tratan sobre los constructores de esos originales/modelos.

Gran parte de la confusión que se produce cuando se trata sobre los lenguajes de modelización y la modelización en general se debe a que los modelizadores, que como hemos indicado son principalmente humanos, necesitan usar un lenguaje para formular las sentencias de un modelo. Por este motivo, las expresiones del lenguaje elegido para expresar estas sentencias se confunden a menudo con el modelo semántico asociado con ellas a través de la correspondencia semántica. Así por ejemplo, no sorprende que los diseñadores de software se refieran a los diagramas de clases UML como modelos. La clave para no cometer este error es ser consciente de que un lenguaje de modelización no es otra cosa que un lenguaje cuyas expresiones no son modelos en sí mismas sino simplemente expresiones. Estas expresiones tienen una correspondencia con modelos semánticos (como en cualquier otro lenguaje), pero al mismo tiempo son expresiones concretas (especificaciones de las sentencias) del modelo con el que se corresponden (el que especifican). En definitiva, la confusión aparece cuando se toma el significante (la

expresión) en vez del significado (el modelo).

Nuestra teoría hace hincapié en que los modelos son independientes de cómo son expresados, pero ¿por qué se produce la confusión a la que hemos hecho referencia? En nuestra opinión está relacionada con el hecho de que es habitual que para crear un lenguaje de modelización, y más concretamente su modelo lingüístico asociado, se tome como punto de partida el lado semántico. Esto es debido a que los modelos existen antes de que necesiten ser expresados. Este hecho es importante porque hace que estos modelos de algún modo influyan en la creación del lenguaje de modelización. Pero ¿en virtud de qué mecanismo se produce esa influencia? A continuación proponemos una definición de lenguaje de modelización que ofrece una respuesta a esta pregunta desde la perspectiva de nuestra teoría.

Definición **8.5.** Lenguaje de modelización
Un lenguaje $\mathcal{ML}$ $(\mathcal{LM}_{\mathcal{ML}}, \mathcal{S}em\mathcal{C}_{\mathcal{ML}})$ es un **lenguaje de modelización** si y solo si existe un metamodelo $\mathcal{MM}$ para todos los modelos que participan en $\mathcal{S}em\mathcal{C}_{\mathcal{ML}}$ y existe un homomorfismo entre las características de $\mathcal{MM}$ y las de $\mathcal{LM}_{\mathcal{ML}}$.

De la definición se deduce que un un lenguaje de modelización se crea para expresar un determinado tipo de modelos, que son originales del mismo metamodelo. Por tanto, los modelos que quieren expresarse se estructuran de acuerdo a las características de ese metamodelo (ver apartado 6.1.4) y este hecho puede usarse para construir el modelo lingüístico de un lenguaje de modelización adecuado para expresar esos modelos. Éste es, en nuestra opinión, el mecanismo al que hacíamos referencia anteriormente. Veamos un ejemplo. Considérese un conjunto de modelos orientados a objetos, que son todos ellos originales del metamodelo de la orientación a objetos (en adelante OOMM). Con el fin de construir un modelo lingüístico (OOLM) que permita expresarlos, usaremos el constructor sintético del metamodelo lingüístico $\mathcal{SC}_{\mathcal{LMM}}$ proporcionándole un alfabeto, un vocabulario y un conjunto de reglas léxicas y sintácticas cuyas características lingüísticas deben estar basadas (ser homomórficas) en las características de OOMM (clases, atributos, relaciones entre clases, etc.). Sin embargo, obsérvese que aunque pudiésemos llegar a usar los mismos nombres para las características lingüísticas que para las semánticas, estas características lingüísticas y sus relaciones son estrictamente lingüísticas. Así si OOMM establece que las clases deben tener atributos, entonces en OOLM los símbolos correspondientes a clase y atributo deben estar relacionados por alguna regla que permita expresar que el atributo pertenece a

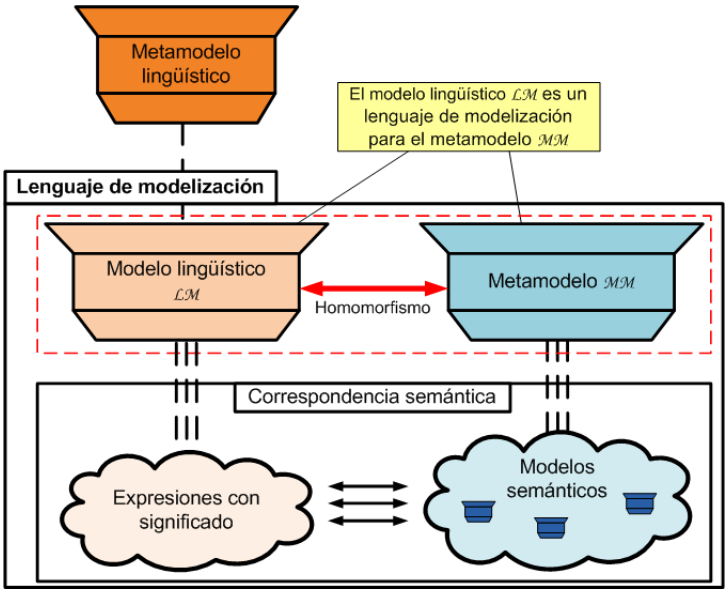


Figura 8.4: Representación gráfica de Lenguaje de modelización

una determinada clase (por ej. en UML esa regla consiste en que una línea horizontal dentro de la caja que representa la clase contiene un elemento que expresa el nombre del atributo).

Igualmente si OOMM establece que las clases pueden relacionarse con otras clases entonces OOLM deberá incluir una regla permitiendo que los símbolos correspondientes a las clases puedan relacionarse entre sí (por ejemplo mediante una línea). Por tanto, este proceso de construcción de OOLM da como resultado un homomorfismo entre las características del lado semántico (OOMM) y las características lingüísticas, y es este homomorfismo el que hace que OOLM sea adecuado para expresar los originales (modelos OO) de OOMM.

Podríamos imaginar que el proceso descrito en el anterior ejemplo debió ser el que siguieron probablemente los autores de las especificaciones de UML para la construcción de este lenguaje de modelización. Evidentemente, un mismo metamodelo puede estar relacionado con diferentes modelos lingüísticos, como por ejemplo UML y el lenguaje Object-Z (los propios autores de las especificaciones de UML resaltan este hecho, ver (OMG, 2011b)[Nota al pie en Pág. 11].

Es importante resaltar que la característica distintiva de los lenguajes de modelización que los diferencia del resto de los lenguajes es la existencia de un metamodelo en el lado semántico y de un homomorfismo entre el metamodelo de los modelos semánticos del lenguaje y su modelo lingüístico. Debe tenerse en cuenta que para algunos lenguajes de modelización sólo es un homomorfismo puesto que

algunas características de $\mathcal{MM}$ no pueden ser expresadas por las expresiones (originales) de $\mathcal{LM}_{\mathcal{ML}}$.

Así por ejemplo, (Lloret, 1998) propone un lenguaje de modelización para los modelos E/R que no puede expresar la cardinalidad en las relaciones. Sin embargo, en otras ocasiones, cuando todas las características de $\mathcal{MM}$ pueden ser expresadas, el homomorfismo pasa a ser un isomorfismo. En este caso aumentan las posibilidades de que las expresiones del lenguaje de modelización y los modelos semánticos puedan ser confundidos entre sí al ser ambos isomórficos. Así, continuando con el ejemplo anterior, para el caso concreto de UML, vemos que un modelo orientado a objetos puede mapearse a una expresión UML puesto que las características lingüísticas de las expresiones de UML (expresión *class*, expresión member, etc.) y las relaciones entre ellas (definidas mediante las reglas de UML) también existen en el lado semántico, es decir en el metamodelo de la orientación a objetos. Este isomorfismo es precisamente la razón por la que los humanos pasamos sin ser conscientes de ello del lado lingüístico al lado semántico atribuyendo a cada característica lingüística una semántica que no tiene por sí misma: la semántica de su correspondiente característica isomórfica. Así por ejemplo, desde un punto de vista estrictamente lingüístico cuando un ingeniero de software dibuja un símbolo de actor en un diagrama de casos de uso de UML, este símbolo no es el responsable de la ejecución de una determinada acción en el sistema diseñado sino que es simplemente un símbolo asociado por una flecha a una elipse que está expresando el caso de uso. Las características lingüísticas (expresión actor, expresión caso de uso, etc.) carecen de semántica. Esa transferencia de semántica hacia el lado lingüístico hace que la comprensión de los metamodelos sea complicada. Los metamodelos se confunden a menudo, e incluso se mezclan completamente con los lenguajes de modelización. Por ejemplo, desde nuestro punto de vista, no sería correcto decir que la sintaxis abstracta de UML es un metamodelo de UML o que UML es un metamodelo: como su propio nombre indica, UML es justamente un lenguaje de modelización. Los lenguajes de modelización homomórficos son, por el contrario, incompletos y pueden dar lugar a muchas ambigüedades porque el intérprete de sus expresiones no puede encontrar las correspondencias que faltan (y muchas veces necesita) con las características de los modelos semánticos (orientados a objetos en el caso de UML) que no son mapeadas homomórficamente.

A modo de resumen, podemos afirmar que un metamodelo no es el modelo de

un lenguaje de modelización, sino más bien *el metamodelo de los modelos semánticos que tienen una correspondencia con las expresiones del modelo lingüístico del lenguaje de modelización*. Nuestra contribución es separar claramente ambos mundos (lingüístico y semántico) e identificar el mecanismo por el que los humanos los mezclamos y confundimos, permitiéndonos identificar la razón por la que un metamodelo es a menudo confundido con un modelo de un lenguaje.

8.4.4. Metalenguaje

Otro término que aparece a menudo en relación con la modelización y metamodelización es el de metalenguaje (ver por ej. (Aß mann et al., 2006)). Desde nuestro punto de vista un metalenguaje es un tipo especial de lenguaje en el que todos los modelos semánticos que aparecen en el lado semántico de la correspondencia semántica son modelos lingüísticos. Dicho de otro modo, en un metalenguaje toda expresión con significado se mapea a través de la correspondencia semántica con un modelo semántico que a su vez es un modelo lingüístico.

DEFINICIÓN **8.6.** Metalenguaje
Un lenguaje $\mathcal{M}t\mathcal{L}$ $(\mathcal{LM}_{\mathcal{M}t\mathcal{L}}, \mathcal{S}em\mathcal{C}_{\mathcal{M}t\mathcal{L}})$ es un **metalenguaje** $\iff \forall m \in \mathcal{SM}$, $m \in \mathcal{O}_{\mathcal{LMM}}$, siendo el conjunto de todos los modelos semánticos que participan en $\mathcal{S}em\mathcal{C}_{\mathcal{M}t\mathcal{L}}$.

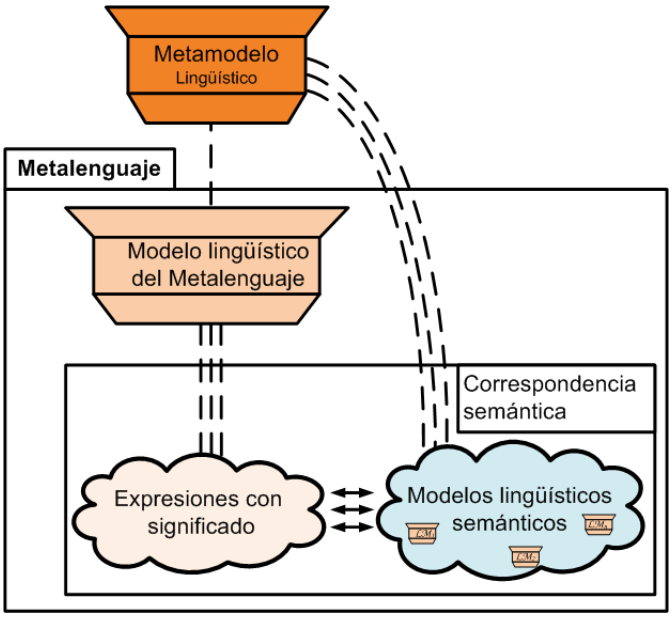


Figura 8.5: Representación gráfica de Metalenguaje

La especificación de XML-Schema (Biron et al., 2004) es un ejemplo de metalenguaje. Una determinada expresión de XML-Schema es un documento concreto que define las restricciones sobre las etiquetas, cardinalidad y otras reglas de un determinado tipo de documentos XML. Por ejemplo, el documento XML Schema correspondiente a MathML contiene las reglas para los documentos XML conformes a este estándar XML sobre matemáticas.

Desde un punto de vista semántico un determinado XML Schema, (por ej. el mencionado correspondiente a MathML), es un modelo lingüístico ($\mathcal{LM}_{MathML}$ en el caso de MathML) que especifica (modeliza) un tipo de documentos XML (en el ejemplo documentos XML matemáticos). Por tanto, cada expresión de XML Schema se corresponde con su modelo lingüístico de XML Schema a traves de la correspondencia semántica del lenguaje de especificación de XML-Schema, por lo que XML-Schema es un metalenguaje según la Definición 8.6. Es interesante observar que puesto que XML-Schema es una concreción de XML, su modelo lingüístico es un submodelo del modelo lingüístico de XML (ver apartado 8.2.3) y al mismo tiempo es un metalenguaje para otros lenguajes basados en XML. Por otro lado, XML-Schema es también un metalenguaje reflexivo puesto que XML-Schema puede ser expresado por sí mismo. Sin embargo, a diferencia de la creencia general, en nuestra opinión a tenor de las definiciones proporcionadas, XML no es un metalenguaje puesto que según la teoría expuesta no es cierto que todo modelo que participe en su correspondencia semántica debe ser un modelo lingüístico. Por tanto, es evidente que quienes califican a XML como un metalenguaje atribuyen a este término un significado muy diferente al expresado por nosotros. Otro ejemplo clásico de metalenguaje es EBNF, cuyas expresiones se corresponden por ejemplo con los modelos lingüísticos de Java u otros lenguajes de programación.

Puesto que en un metalenguaje todos los modelos semánticos son modelos lingüísticos, su lado semántico se rige por un un metamodelo, que no es otro que el metamodelo lingüístico $\mathcal{LMM}$. En algunas ocasiones incluso sería posible que el lado semántico de los modelos lingüísticos tenga algunas características comunes que nos ayuden a determinar un metamodelo más concreto que $\mathcal{LMM}$ (el cual sería un submodelo de $\mathcal{LMM}$). Lo que resulta interesante en este caso es que, en virtud de lo descrito en el apartado 8.4.3, si pudiese encontrarse un homomorfismo entre el metamodelo del lado semántico y el modelo lingüístico del metalenguaje, podríamos concluir que ese metalenguaje también sería un lenguaje de modelización.

8.5. Resumen y conclusiones

Nuestra propuesta sobre lenguaje es similar en lo esencial a otros estudios sobre esta materia que también separan y al mismo tiempo relacionan los aspectos lingüísticos y semánticos de un lenguaje (Bloom and Keil, 2001; Chen et al., 2005; Gonzalez-Perez and Henderson-Sellers, 2007; Harel and Rumpe, 2000; Mu et al., 2010; Nowak et al., 2002; Sowa, 2000; Ullmann, 1962; Visser, 2008). Sin embargo, difiere en que está centrada en el modelo (y el metamodelo) asociado a un lenguaje. Este enfoque nos ha permitido no sólo hablar sobre las propiedades lingüísticas de los lenguajes, sino que también, gracias a los constructores de un modelo hemos podido explicar como se estudian (analizan) y crean (sintetizan). Aunque varios autores afirman que un lenguaje es un conjunto de expresiones o un sistema de símbolos, nosotros concluimos que en esos estudios el término *lenguaje* está refiriéndose al conjunto de originales (expresiones) de un modelo que hemos denominado modelo lingüístico. Este modelo junto con una correspondencia semántica es lo que nosotros hemos denominamos lenguaje. Otra diferencia es que algunos autores usan el término *gramática* (grammar) para referirse a lo que hemos denominado modelo lingüístico (ver por ejemplo Favre and Nguyen (2004)).

Coincidimos con Favre (2005) cuando afirma que no debe confundirse un lenguaje con los modelos que lo describen, pero nos diferenciamos en cómo entendemos el concepto de lenguaje y el modelo (en singular) que lo describe. Mientras que para Favre un lenguaje es un conjunto de sistemas y una gramática es un modelo de un lenguaje, para nosotros un lenguaje es una entidad caracterizada por un modelo lingüístico (gramática) y una correspondencia semántica.

Los términos sintaxis abstracta y sintaxis concreta merecen una especial atención en esta sección puesto que muchos autores las utilizan cuando debaten sobre los lenguajes (Atkinson and Kühne, 2003; Baar, 2006; Basin et al., 2006; Bézivin et al., 2005; Boronat et al., 2007; Feilkas, 2006; Frankel, 2003; Gargantini et al., 2009; Guizzardi, 2007; Harel and Rumpe, 2000; Kelly and Tolvanen, 2008; Kramer, 2007; Kühne, 2006a; Kurtev, 2007; Lodderstedt et al., 2002; Mens and Van Gorp, 2006; Ober and Prinz, 2006; OMG, 2011b,c; Overbeek, 2006; Seidewitz, 2003; Selic, 2005; Shan and Zhu, 2008; Wagner et al., 2005; Xia and Glinz, 2003). En general, podemos decir que estos dos aspectos son tenidos en cuenta en nuestra definición de $\mathcal{V}_{\mathcal{LMM}}$ (Definición 8.1). Sin embargo, preferimos no utilizar esta terminología. Aunque nuestra definición se refiere principalmente a los aspectos

notacionales o sintaxis concreta (alfabeto, lexemas y las reglas sintácticas para la generación de expresiones), también hemos mencionado que las características lingüísticas (ver apartado 8.2.4) son la verdadera base que permite la definición de las reglas de un modelo lingüístico. Estas características lingüísticas (junto con sus relaciones) son lo que otros autores han denominado sintaxis abstracta. Desde nuestra perspectiva, además queda claro que esa sintaxis abstracta es un término que se sitúa en el plano conceptual, aunque los conceptos que la forman son de naturaleza puramente lingüística.

Esta última cuestión es un tema que ha sido fuente de no pocos problemas de interpretación y de debate en la literatura (véase por ejemplo (Favre, 2005; Kühne, 2006a; Kurtev, 2007)). A nuestro juicio, las características lingüísticas de un modelo lingüístico, o si se quiere su sintaxis abstracta, están desprovistas de semántica. Ésta es una cuestión bastante difícil de explicar debido a que a los humanos nos resulta muy complicado separar el significado de una característica lingüística cuando nos referimos a ella por su nombre. Así por ejemplo, si decimos que algo es un *"operando"* en un modelo lingüístico para expresiones aritméticas, inconscientemente estamos asociándolo con un número que tiene un valor y que puede ser operado con otro número para dar un resultado. Por tanto, debemos hacer un esfuerzo para entender que esa característica que estamos llamando *"operando"*, podría haber sido nombrada como *"xyz"* en el modelo lingüístico, y que, desde un punto de vista lingüístico, lo único que importa es que aparece antes o después de otro elemento (lingüístico) que llamamos *"operador"* (y que podríamos haber llamado por ejemplo *"zkl"*). La confusión y malinterpretación es aún más grave en el caso de los lenguajes de modelización, que es donde nuestra teoría se diferencia aún más de otros autores. Nuestra contribución aquí es separar con claridad los aspectos lingüísticos y semánticos de un lenguaje de modelización, así como ilustrar el mecanismo por el cual los humanos los confunden. De este modo se ofrece una explicación de por qué diversos autores identifican metamodelo con el modelo de un lenguaje de modelización (ver RCM de metamodelo en la sección 4.1.2) o sintaxis abstracta con metamodelo (Bézivin et al., 2005; Feilkas, 2006; Gargantini et al., 2009; Visser, 2008), o como (Favre, 2005) identifica *lenguaje de modelización* con un *conjunto de modelos* (ver RCM de Lenguaje de modelización en el apartado 4.1.3), o incluso simplemente identifican una expresión de un *lenguaje de modelización* con el modelo (semántico) al que expresa. La explicación a todas estas confusiones es, según nuestra definición de lenguaje de modelización,

el homomorfismo existente entre las características lingüísticas y las del metamodelo semántico. Por ejemplo, volviendo al caso de la orientación a objetos, la característica «class» del metamodelo de la orientación a objetos (OOMM) es puramente semántica y muy diferente de la característica lingüística *class* que aparece en las sentencias del modelo lingüístico de los Diagramas de clase de UML (nótese la diferente tipografía que empleamos en este ejemplo para distinguir ambos términos). Efectivamente «class» y *class* pertenecen a dimensiones diferentes. Sin embargo, y éste es el punto principal, UML no sólo usa el mismo término (*class*) para denominar clases de diferentes niveles como mostramos en la sección 4.2.2, sino que también usa la misma palabra para referirse a «class» y *class* generando una considerable confusión (Rodriguez-Priego et al., 2010). Tal confusión ha llevado a algunos autores (véase por ejemplo Kühne (2006a)) a proponer dos vistas de UML: la ontológica y la lingüística, cuando en realidad sólo existe la visión lingüística de UML (por algo, como dicen sus siglas, es un lenguaje) y esa segunda vista ontológica no es parte de UML sino que se corresponde con los modelos semánticos orientados a objetos expresados mediante UML en virtud de su correspondencia semántica.

Una visión diferente es la de Seidewitz (2003), que define un lenguaje de modelización como el empleado para expresar las sentencias de modelos de algún sistema bajo estudio. De acuerdo con este punto de vista, el lenguaje natural podría ser considerado un lenguaje de modelización, aunque muchas de las expresiones generadas por él nunca se correspondan con sentencias de cualquier modelo por ser *meaningless*. Nuestra propuesta es diferente y plantea unos requisitos más rigurosos.

Capítulo 9

Pragmática

En los capítulos anteriores hemos tratado los principales conceptos de modelización (modelo, original, lenguaje, etc.) y sus relaciones (submodelización, metamodelización, etc.) desde un punto de vista teórico. En este capítulo abordaremos el aspecto práctico de la modelización. Los modelos son resultado de procesos cognitivos humanos que nos permiten construirlos con una finalidad. En cierta medida, parte de ese aspecto pragmático incluye el aspecto práctico del método científico en el que está basada esencialmente nuestra teoría. Efectivamente, como ya indicamos en el apartado 5.2.1, el científico realiza una investigación partiendo de la formulación de un problema y de una finalidad concreta. De manera similar, el modelizador construye un modelo para conseguir una finalidad determinada. La propia definición de modelo ya permite intuir que la motivación del modelizador está basada en los procesos de análisis, analogía o síntesis. Por ejemplo, un modelizador puede crear un modelo sintético de mesa con la finalidad última de construir mesas físicas o como base para pintar cuadros sobre mesas.

Veremos a continuación que el aspecto pragmático de la modelización está relacionado con la comunicación y con los roles que adopta cada uno de los sujetos que intervienen en el uso de un modelo.

9.1. Roles de modelización

Hasta el momento hemos utilizado el término *modelizador* para referirnos a aquel sujeto, generalmente humano, que crea, usa y trabaja en general con modelos y que, en definitiva, es el responsable de realizar los diferentes procesos asociados con los modelos que hemos definido hasta el momento: validación, análisis, ana-

logía y síntesis. Sin embargo, aunque nos hayamos referido a este sujeto como un único actor, no nos estamos refiriendo necesariamente a un único sujeto físico sino que se trata de una abstracción que agrupa los diferentes roles que pueden ser desempeñados por los sujetos que intervienen en los procesos de modelización.

Rol	**Acción**	**Entrada**	**Resultado**	**Mecanismo**
Analista (creador)	Crear un modelo	Muestra de originales percibidos	Un modelo	Constructor analítico
Analogista (creador)	Crear un modelo	Modelo de referencia	Un modelo	Constructor analógico
Validador	Validar el original de un modelo	Una entidad	Verdadero si la entidad es un original del modelo y falso en caso contrario	Función de validación
Sintetizador	Crear originales a partir del modelo	Un conjunto de semillas	Originales concebidos	Constructor sintético
Transmisor	Expresar un modelo utilizando las expresiones de un lenguaje	Un modelo	Una expresión	Lenguaje
Intérprete	Interpretar un modelo que ha sido expresado a través de las expresiones de un lenguaje	Una expresión	Un modelo	Lenguaje
Usuario	Usar un modelo y realizar las acciones necesarias para alcanzar una finalidad	Un modelo, entidades externas	Lograr una finalidad	

Tabla 9.1: Los roles de la modelización

Según nuestro conocimiento éste es un tema que apenas ha sido tratado en la literatura sobre modelización si lo comparamos con los debates sobre sus fundamentos teóricos y que se ha centrado en los roles más básicos. Así por ejemplo, Overbeek (2006) diferencia dos actores: el modelizador, que crea el modelo y el intérprete que es quien lo entiende con el propósito de usarlo. Sin embargo, de la teoría expuesta podemos observar que existen más roles de modelización y algo más complejos, siendo posible así mismo que un modelizador pueda ejercer diferentes roles en diferentes momentos. La tabla 9.1 recopila los diferentes roles de modelización que podemos distinguir en base a nuestra teoría: analista,

analogista[1], validador, sintetizador, transmisor, intérprete y usuario. La primera columna indica el nombre del rol. A continuación se indica la acción que realiza el sujeto que ejerce dicho rol. Las dos siguientes columnas indican qué información de entrada utiliza para ejercer dicha acción y el resultado obtenido al realizar la acción indicada. Finalmente se indica el mecanismo de modelización empleado, que se corresponde con alguno de los descritos en los capítulos anteriores. Teniendo en cuenta estos roles pueden producirse diversos escenarios. A continuación describimos dos escenarios genéricos que ilustran cómo usan los modelizadores estos roles.

9.2. Modelizador independiente

Este escenario se corresponde con aquél en el que un sujeto usa modelos de manera autónoma sin comunicarlos a otros sujetos (personas o máquinas). En esta situación, aunque el sujeto puede utilizar un lenguaje para expresar el modelo, y así permitirle recordarlo o como ayuda para su estudio, es evidente que en general no es necesario el uso de un lenguaje para la utilización de estos modelos (que denominamos *modelos no expresados* en la sección 8.1). En la misma sección 8.1 podemos encontrar algunos ejemplos de este tipo de modelos (ejemplos de modelo de cara para reconocimiento facial y modelo de nuestra visión del mundo). En ambos casos el sujeto usa el modelo, sin expresarlo, para conseguir un objetivo personal (reconocer una cara o desenvolverse normalmente en su vida diaria) ejerciendo todos los roles indicados en la tabla 9.1 (salvo transmisor e intérprete), puesto que él mismo analiza, sintetiza, valida y usa sus modelos.

9.3. Modelizador social

El caso más habitual es en el que dos o más modelizadores intercambian modelos para conseguir una determinada finalidad (la misma o diferente para cada uno de ellos). Para poder realizar ese intercambio deben compartir el conocimiento de un determinado lenguaje. Ese conocimiento de un lenguaje implica el de su modelo lingüístico y de un conjunto de correspondencias semánticas, parte de las cuales

[1]Aunque sí existe el término en inglés *analogist* para este concepto, no hay un término equivalente en español. Proponemos la utilización del término *analogista* similar al término en inglés.

son comunes a todos los que conocen y usan un lenguaje (*relaciones semánticas colectivas* de un lenguaje). Además, si la comunicación tiene éxito, los modelizadores habrán ampliado sus respectivos conjuntos de correspondencias semánticas con al menos una (la del modelo expresado y comunicado), aunque como veremos a continuación también podrán comunicarse otras correspondencias.

En virtud de la Definición 8.4 podremos decir que cada uno de los modelizadores que interviene en la comunicación, aunque comparten el mismo modelo lingüístico, no comparten el lenguaje completo sino un sublenguaje del mismo. De hecho ningún humano conoce y utiliza la totalidad de un lenguaje, no sólo por el elevado número de sus relaciones semánticas, sino porque los lenguajes naturales se caracterizan por su continua evolución (tanto en el modelo lingüístico como en su correspondencia semántica).

9.3.1. El proceso de la comunicación de modelos

El proceso de comunicación en un escenario de modelizador social comienza con las expresiones. Efectivamente, el modelizador con el rol de transmisor elige un lenguaje para expresar el modelo, obtiene una expresión (mediante el constructor sintético del modelo lingüístico) correspondiente al modelo (correspondencia semántica) y utilizando algún medio de comunicación transmite esa expresión al sujeto con el rol de intérprete. No debe perderse de vista que el modelo reside físicamente (mentalmente en el caso de los humanos) en las entidades que realizan las acciones definidas en la tabla 9.1. Las únicas entidades físicas que permiten el intercambio de modelos, y que los modelizadores utilizan como apoyo, son los medios de comunicación (papel, correo-e, voz, etc) mediante los cuales los modelizadores trasladan a un medio físico las expresiones sintetizadas a partir del modelo lingüístico.

En un primer momento puede parecer que en una comunicación de modelos los únicos roles que ejercen los modelizadores son los de transmisor e intérprete, sin embargo los mecanismos por los que otro humano (o una máquina) llegan a *comprender* el modelo son complejos y en ellos intervienen a su vez otros procesos (cognitivos en el caso humano) algunos de los cuales se corresponden con los roles indicados en la tabla 9.1. El objetivo en este apartado es responder a la siguiente cuestión: ¿cómo puede el transmisor comunicar un modelo (con toda su complejidad) al intérprete simplemente con las expresiones de un lenguaje? Mostraremos a continuación como la respuesta a esta pregunta desde el enfoque

de nuestra teoría sobre modelización ayuda a comprender el importante papel que juega la modelización en la comunicación mediante un lenguaje.

La primera consideración es que, según nuestro enfoque, mediante el lenguaje se comunican no simplemente conceptos (como imágenes mentales), sino modelos. En efecto, en nuestra opinión, la comunicación de una imagen mental de un modelizador a otro (si eso fuese posible directamente) no sería efectiva si esa imagen no fuese la de un modelo. Así por ejemplo, si un transmisor comunica la expresión "casa", el intérprete sólo entenderá realmente lo que es una «casa» si es capaz de reconocer casas (validar), refinar las sentencias sobre casa (análisis) observando muestras o incluso *concebir* nuevas casas (síntesis). Por tanto, para que el intérprete pueda recibir un modelo comunicado por el transmisor debe obtener el modelo completo (su protomodelo y sus constructores). El problema es que el transmisor no puede transmitir directamente sus imágenes mentales de protomodelo y constructores y por tanto debe conseguir que el intérprete las *recree* en su mente a partir de las expresiones del lenguaje utilizado.

A continuación describimos cuáles son los mecanismos que permiten este complejo proceso, teniéndose en cuenta que estos mecanismos no son excluyentes (dos o más de ellos pueden ser aplicados simultáneamente):

Por correspondencia directa

El primer mecanismo es trivial: si el intérprete ya conoce la expresión, tendrá una correspondencia semántica o si tiene varias (sinonimia) podrá elegir la que sea más adecuada según el contexto y la finalidad. En ese caso, contrastará que su correspondiente modelo coincide o es equivalente con el del transmisor y por tanto lo podrá usar directamente. El transmisor únicamente ha comunicado una parte de la correspondencia, la relativa a las expresiones.

Por descripción

Para que el intérprete pueda comprender el modelo, el transmisor elige un lenguaje para expresar las sentencias del modelo y le transmite esas expresiones. Como indicamos en el apartado 6.1.4 las sentencias hacen referencia a las características del modelo y sus relaciones. Por otro lado, si el lenguaje es el apropiado las características del modelo lingüístico serán homomórficas con las características del modelo (ver apartado 8.3.4) y las subexpresiones correspondientes a esas características lingüísticas tendrán su correspondencia semántica con los modelos

correspondientes a las características en sí, y lo mismo ocurrirá para las expresiones correspondientes a las relaciones.

Por ejemplo, si expresamos una de las sentencias del modelo de mesa con la expresión

"tiene un tablero soportado por las patas"

el intérprete deberá reconocer las siguientes correspondencias semánticas:

"tiene"	→	Modelo de *tener*
"un tablero"	→	Modelo de *tablero*
"soportado por"	→	Modelo de *soportar*
"las patas"	→	Modelo de *pata*

En este ejemplo las características son *tablero* y *pata* y la relación entre ambas es *soportar* (estar en encima de).

El conocimiento compartido del modelo lingüistico (en este caso el del lenguaje español) ha permitido al intérprete identificar más fácilmente cuáles son las expresiones relativas a objetos (tablero, pata) y a relaciones (soportar). Por otro lado, le ha permitido interpretar que el sujeto de "tiene" es la entidad a la que se refiere la expresión completa. Lógicamente se trata de un proceso recursivo, puesto que la comunicación de cada uno de esos modelos podrá usar a su vez alguno de los mecanismos que se describen en esta sección. Es evidente que el intérprete no podrá comprender el modelo de mesa si no comprende previamente el modelo de tablero o pata. La utilización de los modelos de las características para obtener el modelo implica que, tal como indicamos en la sección 7.3, el modelo transmitido se obtiene con la ayuda de los modelos característicos. Así mismo, el uso de un lenguaje, con sus correspondencias semánticas colectivas permite que este proceso recursivo también pueda utilizar otros mecanismos como el de ir reutilizando las *Correspondencias directas* que el intérprete ya conoce o va aprendiendo en cada comunicación.

Respecto a los constructores, la forma más directa de transmitirlos es expresar a su vez la descripción de los mismos mediante un lenguaje. Esta acción a su vez implica transmitir el modelo del constructor mediante el proceso recursivo al que acabamos de hacer referencia en el caso del protomodelo. Cuando en la vida cotidiana una persona explica a otra cómo se obtiene la descripción de algo o cómo se obtiene una particularización de algo se está realizando una acción similar a la que nos referimos en el caso de los constructores de un modelo.

Nótese que si comparamos el proceso de recreación del modelo por parte del

intérprete con los procesos de creación de modelos descritos por la teoría (constructor analítico, analógico, constructor sintético de metamodelo) no se corresponde con ninguno de ellos, puesto que el intérprete no parte de una muestra de originales ni de un modelo de referencia. Se trata de otro tipo de constructor que podríamos llamar constructor interpretativo y que está basado en el proceso que hemos expuesto, pero que no impide que el modelo interpretado sea igualmente analítico, analógico, sintético o completo.

Por Analogía inducida por el lenguaje

El transmisor envía igualmente las sentencias del modelo al intérprete, pero en este caso las expresiones, aunque no tienen correspondencia semántica con el modelo que pretende comunicar el transmisor, sí tienen una correspondencia con otros análogos, a partir de los cuales el intérprete puede obtener las sentencias del modelo. Lo que ocurre en este caso es que el lenguaje ha permitido al intérprete utilizar un modelo de referencia que ya conocía y mediante el rol de *analogista* deducir el modelo transmitido.

Veamos un ejemplo. Imaginemos que el transmisor expresa las sentencias del modelo de pila (stack), entendido en el contexto del desarrollo de software. Si el intérprete no conoce ese modelo pero sí el modelo genérico de pila de objetos físicos (por ej. una pila de platos) esto le podrá ayudar a entender por analogía las sentencias del modelo de pila software. Así, la característica expresada como "cima" de la pila (top) podrá interpretarla más fácilmente a partir del modelo de *cima* en una pila de objetos físicos, que si no conociese previamente ese modelo de referencia.

Por análisis o analogía inducidas

Otro modo de obtener las sentencias del modelo correspondiente a la expresión comunicada es inducir en el intérprete un proceso de análisis o analogía para que las obtenga él mismo. Con este propósito el transmisor podría indicarle al intérprete cuáles son los originales físicos o de referencia (análogos) asociados al dominio del modelo para que pueda observarlos y asumiendo los roles de intérprete-analista/analogista obtener el protomodelo.

El transmisor también puede *ayudar* al intérprete en este proceso indicándole los originales más idóneos como muestra o incluso un prototipo del modelo que, como indicábamos en el apartado 6.2.1, permitiría al intérprete comprender el ho-

momorfismo existente entre las características del modelo y los correspondientes valores en el prototipo y a partir de ahí obtener las sentencias del modelo.

Obsérvese que este proceso de análisis o analogía es posible bien porque el transmisor le ha comunicado el constructor analítico (por ejemplo, por descripción) o porque el intérprete de manera autónoma ha construido su propio constructor analítico o analógico por inducción a partir de la muestra o prototipo proporcionada por el transmisor. En este último caso, el intérprete obtiene el protomodelo y el constructor analítico o analógico al mismo tiempo.

Por Síntesis inducida por metamodelo

Si transmisor e intérprete comparten el conocimiento del metamodelo del modelo transmitido entonces el transmisor puede inducir la síntesis del protomodelo a partir del metamodelo común. En este caso también obtendría los constructores puesto que como vimos en el apartado 7.2.1 existe una relación muy estrecha entre el constructor sintético del metamodelo y los constructores de sus originales/modelos, especialmente si son analíticos o analógicos. Como indicamos en el apartado 8.4.3 en este caso el lenguaje utilizado por el transmisor es lo que se conoce como lenguaje de modelización.

Veamos un ejemplo. Supongamos el caso de la comunicación de un modelo orientado a objetos realizado para el desarrollo de una aplicación informática.

En primer lugar el modelizador con el rol de analista realiza el análisis del dominio de la aplicación y crea el modelo a través de su constructor analítico. Durante este estadio el modelizador no sólo ha creado el protomodelo por la observación de una muestra de originales sino que también ha utilizado el constructor sintético del metamodelo de la orientación a objetos, el cual le ha llevado a construir sentencias basándose en clases que representan objetos, atributos, métodos, etc.

Una vez construido el modelo, el modelizador ejerciendo ahora el rol de transmisor, elige un lenguaje de modelización como por ejemplo UML, para expresar las sentencias del modelo. Utilizando el modelo lingüístico de UML el transmisor primero imagina y después escribe una expresión en UML (por ej. un diagrama de clases) que se corresponde con el modelo (relación semántica). Este diagrama de clases es una expresión compleja que contiene muchas de las características lingüísticas del modelo lingüístico de UML. Éste es adecuado para expresar un modelo orientado a objetos porque existe un homomorfismo entre las caracte-

rísticas del metamodelo de la orientación a objetos y el modelo lingüístico de UML.

A continuación el transmisor envía el modelo al modelizador con el rol de intérprete (por ejemplo al programador que va a desarrollar la aplicación) utilizando algún medio de comunicación. El intérprete lee el diagrama de clases y realiza el proceso inverso: primero lo valida como una expresión del modelo lingüístico de UML y después reconstruye el modelo en su mente. Habitualmente puede producirse un diálogo entre transmisor e intérprete para aclarar o refinar el diagrama.

En este ejemplo para la recreación del modelo por parte del intérprete en el mismo modelo o al menos en uno equivalente, intervienen tres factores: 1) Transmisor e intérprete comparten el mismo lenguaje (de modelización) y por tanto utilizan el mismo modelo lingüístico junto con un conjunto suficientemente amplio de relaciones semánticas que son la base para la comprensión y transmisión del nuevo modelo. Así por ejemplo, tal como ya hemos mostrado el intérprete comprende que el atributo "fechaNacimiento" de la clase "Persona" es una fecha porque conoce de antemano el modelo semántico de fecha, 2) la existencia de un homomorfismo entre las características del metamodelo de la orientación a objetos y las características del modelo lingüistico, y 3) el metamodelo de la orientación a objetos, cuyo conocimiento comparten transmisor e intérprete, que permite que éste último pueda recrear las sentencias del modelo usando el constructor sintético de dicho metamodelo.

Por Deducción de sub/supermodelo

Otro caso interesante es aquel en el que el transmisor indica al intérprete que el modelo que quiere transmitir es un submodelo o supermodelo de otro, en una de las sentencias del modelo transmitido. Si el intérprete ya conoce el submodelo o supermodelo al que se refiere, puede deducir las sentencias del modelo generalizando o particularizando respectivamente de las del submodelo o supermodelo conocido, usando para deducir la diferencia entre los modelos conocidos alguno de los mecanismos anteriormente expuestos.

9.4. Pragmática de los modelos software

Hemos mencionado en diferentes ocasiones que el modelizador es típicamente humano, puesto que el origen de la modelización y de su comunicación mediante el lenguaje es humano. Alguno de los roles indicados, especialmente el de creador de modelos es actualmente más difícil de ejercer por una máquina (una excepción es el caso de las herramientas de ingeniería inversa). Sin embargo, sí es posible comunicar modelos a una máquina mediante los lenguajes de programación para que los interprete y los use. Los lenguajes de programación son un caso especial de lenguajes de modelización (Guéhéneuc et al., 2002) y por tanto siempre existe un metamodelo homomórfico con su modelo lingüistico. Así por ejemplo, el lenguaje Java está relacionado homomórficamente con el metamodelo de la orientación a objetos como también lo está UML.

Una de las particularidades de los modelos interpretados por máquinas es que comúnmente son sintéticos dada la dificultad por parte de una máquina de realizar un proceso analítico percibiendo muestras de originales para inducir las sentencias del modelo y poder luego obtener los constructores, como lo haría un humano observando la realidad a través de sus sentidos, analizándola y obteniendo un modelo analítico. El método habitual en el caso de comunicación con máquinas es, por el contrario, el de transmitir al sistema software el modelo del constructor sintético correspondiente al modelo transmitido en forma de sentencias de un lenguaje de programación que pueda ser interpretado por el sistema. Por tanto, para el caso de los sistemas software, tanto el protomodelo como los constructores de un modelo realizado por un transmisor son expresados mediante lenguajes de programación.

El resultado de la interpretación de lo transmitido por parte del sistema informático es la implementación de la función de validación y de los constructores (generalmente sintéticos) mediante software. Es interesante observar, tal como indicamos en la tabla 6.1 a propósito de la naturaleza física y abstracta de los modelos, que la *ejecución* de los constructores da como resultado imágenes digitales análogas a las imágenes mentales en el caso de los humanos.

9.5. Utilización de modelos

El último paso en un proceso de comunicación es el uso del modelo, una vez que el intérprete ha recibido la comunicación del modelo por parte del transmisor. Debe tenerse en cuenta que la utilización por parte del intérprete de los mecanismos expuestos no tiene necesariamente como objetivo que ambos, transmisor e intérprete, tengan el mismo modelo. En realidad, el caso más común es justamente el contrario: es difícil que el modelo transmitido y el modelo interpretado sean el mismo por diferentes razones:

- El intérprete, salvo que entienda y siga literalmente las instrucciones del transmisor, creará por su cuenta constructores diferentes e incluso sentencias diferentes (aunque las funciones de validación sean iguales). En definitiva, la situación más probable es que los modelos transmitido e interpretado, más que iguales, sean equivalentes.
- El intérprete no tiene que crear los mismos constructores que el transmisor, es decir, una vez obtenida la función de validación de un modelo transmitido analítico, el intérprete puede crear (por ej. con el mecanismo de Síntesis inducida por metamodelo) un constructor sintético y descartar el analítico o puede obtener un modelo completo.
- Las finalidades del transmisor y el intérprete no tienen por qué ser las mismas. En el anterior ejemplo de comunicación de un modelo orientado a objetos el modelizador-analista (transmisor) crea y utiliza el modelo para obtener una descripción de un sistema real basándose en la orientación a objetos, mientras que el intérprete lo utiliza para desarrollar una aplicación informática. En este caso además están interesados en constructores distintos (el primero en el analítico y el segundo en el sintético).

9.6. Existencia y destrucción de modelos

Hasta ahora hemos descrito como se crean los modelos, pero cabe hacerse una pregunta que no es tratada generalmente en la literatura: ¿cuál es el ciclo de vida de su existencia? Para responder a esta cuestión hay que volver a prestar atención al origen de los modelos. Hemos indicado que los modelos son creados por un modelizador, generalmente humano. En este capítulo hemos tratado además el tema de la transmisión de modelos por parte de un modelizador a otro (que puede

ser una persona o una máquina) mediante el uso del lenguaje. También hemos visto que el modelo recibido puede ser igual o no al modelo original transmitido.

Podemos afirmar que cuando un modelizador crea un modelo, éste mantiene su existencia con él (en su mente) hasta que la persona deja de existir o simplemente lo olvida. Por otro lado, el modelizador también puede *decidir* que un modelo de su creación *desaparezca al dejarlo en el olvido*, por ejemplo al sustituirlo por otro derivado de éste (ver sección 7.4). En resumen un modelo puede considerarse desaparecido cuando:

- Desaparece el único modelizador que lo posee (que puede ser su creador u otra persona o máquina a quien se lo transmitió)
- El modelizador *lo hace desaparecer* sin solución de continuidad por considerarlo inútil o porque lo *transmuta* en otro modelo (sección 7.4)

Si el modelo no ha sido comunicado mediante un lenguaje entonces desaparecerá para siempre hasta que, probablemente por casualidad, otro modelizador vuelva a crearlo. Dadas las dificultades que pueden existir para transmitir fielmente los constructores del modelo, es muy posible que de un modelo comunicado que se quiera preservar sólo quede un modelo equivalente.

Cuando un modelo es un original de otro modelo (metamodelo), la situación puede ser mejor puesto que aunque el modelo desaparezca (por las circunstancias que hemos mencionado), otro modelizador podría volver a crearlo a partir de su metamodelo, si cuenta con las semillas adecuadas.

Muchos de los modelos creados son un patrimonio que merece ser preservado, aunque sólo quede un modelo equivalente al de su creador. De ahí la importancia de expresar correcta y fielmente los modelos para garantizar su existencia a lo largo del tiempo.

Capítulo 10

Revisión de los problemas de la modelización

En el capítulo 4 describimos con detalle los principales problemas encontrados en nuestro estudio de los fundamentos de la modelización. La teoría que hemos desarrollado descrita en los capítulos anteriores tuvo precisamente como objetivo principal ofrecer una alternativa que permitiese dar respuesta a todas las preguntas planteadas desde un enfoque coherente, consistente y con un punto de vista genérico de la modelización aplicable a cualquier ámbito.

Ha quedado de manifiesto que la modelización es un tema complejo, sujeto a muchos puntos de vista y con multitud de posibilidades de confusión.

En este capítulo repasaremos nuevamente la lista de problemas expuestos para ofrecer qué respuestas ofrece nuestra teoría a cada uno de ellos.

10.1. Dificultades en la comprensión de los conceptos básicos de la modelización

Nuestra respuesta a las confusiones sobre los conceptos básicos de la modelización parte del principio de independencia descrito en la sección 5.1. En virtud de este principio *modelo*, y el original como entidad que modeliza, son los conceptos de primer orden que se definen sin basarse en el resto de conceptos relacionados con la modelización como *lenguaje* o *sistema*.

10.1.1. Modelo

Modelo es una entidad abstracta vinculada a un original a través de un protomodelo y unos constructores, mediante los cuales es posible validar los originales, construir y refinar el protomodelo para que los describa fielmente y crear nuevos originales.

Nuestra definición no necesita de la existencia de un lenguaje ni de un sistema, pero eso no es incompatible con que en la práctica la mayoría de los modelos se expresen en un lenguaje y sus originales sean sistemas.

El RCM de modelo (apartado 4.1.1) muestra la gran cantidad de conceptos relacionados con *modelo*. Todas las definiciones, de un modo un otro, tienen una parte de verdad respecto a qué es un modelo, pero lo que no queda claro es cuales son un requisito para que un modelo pueda llamarse como tal. La teoría expuesta, por el contrario, expone claramente que hay diferentes tipos de modelos (caracterizados por sus constructores). Estos constructores están basados en el método científico que es otro de los principios fundamentales de la teoría. Se han mostrado las sólidas analogías existentes entre los métodos para la investigación científica y los constructores de un modelo.

Llegados a este punto, y a modo de resumen podemos realizar una *definición resumen* de modelo (ver apartado 3.3.6), según la teoría expuesta ayudados por el RCM de modelo que permita recoger y clarificar las principales definiciones encontradas en la literatura:

> Un modelo es una **entidad abstracta** creada por un modelizador (típicamente un humano o un **sistema software**), relacionada con otras entidades abstractas denominadas **originales** si y sólo si son ciertas un **conjunto** de **sentencias** asociadas al modelo (función de **validación**) para esos originales, y el modelo **describe** o **especifica** (o ambos) sus originales a través de un constructor analítico o analógico (descripción) y/o sintético (prescripción) el cual utiliza para este fin **simplificaciones** de los originales.

10.1.2. Metamodelo

Un metamodelo es un tipo especial de modelo. El RCM de metamodelo (apartado 4.1.2) puso de manifiesto que la discusión sobre su significado está en el *secondary layer*. Nuestra teoría se posiciona con quiénes afirman que es un modelo de otros modelos frente a quiénes afirman que es el modelo de un lenguaje

de modelización. La definición de metamodelo propuesta va mucho más allá al profundizar en ciertas características de los metamodelos que son una consecuencia de la propia definición de modelo: el protomodelo de un metamodelo valida también a los constructores de los modelos que son sus originales y además hemos descrito unas relaciones muy interesantes que se producen en algunos casos entre los contructores del metamodelo y los de sus originales.

10.1.3. Lenguaje y Lenguaje de modelización

Nuestro punto de partida para definir qué es un lenguaje de modelización fue igualmente comenzar por la propia definición de lenguaje. Ya vimos en el RCM de lenguaje que la mayoría de los autores estudiados (salvo uno) (ver apartado 4.1.3), definen lenguaje como un conjunto o un sistema de símbolos. Nuevamente, teniendo en cuenta el principio de independencia, nuestra definición se basa en modelo, concretamente en lo que denominamos modelo lingüístico. El dominio de ese modelo lingüístico es lo que gran parte de la literatura llama lenguaje. Para nosotros un lenguaje es una entidad caracterizada por un modelo lingüístico y por una correspondencia semántica. Dicha correspondencia vincula originales del modelo con modelos semánticos.

Nuestra teoría de la modelización tiene la ventaja de que su aplicación al *lenguaje* nos permite entender mejor el origen de alguna de las confusiones que se han producido en su estudio en el contexto de las TIC. Definimos *lenguaje de modelización* a partir de *lenguaje* como un caso particular en el que se produce un homomorfismo entre el metamodelo de sus modelos semánticos y su modelo lingüístico. Esta definición permite explicar, y diferenciar claramente, metamodelo de lenguaje de modelización, haciendo hincapié en que no es que un metamodelo tenga una parte ontológica y otra lingüística (Kühne, 2006a), sino que se trata de cosas muy diferentes aunque relacionadas entre sí.

10.2. Dificultades sobre las relaciones entre modelización y orientación a objetos

10.2.1. El enfoque de OMG Orientado a objetos sobre la modelización

Todos los problemas sobre la elección de un enfoque orientado a objetos para la modelización desaparecen en el momento en el que nuestra teoría no está basada en la orientación a objetos en ningún caso, sino que se fundamenta en la definición de conceptos de modelización puros: (proto)modelo, original, función de validación y constructores. Esto no es obstáculo para que puedan definirse modelos concretos basados en el metamodelo de la orientación a objetos, con sus función de validación y constructores. Lo importante, es que la orientación a objetos no es la base de nuestra teoría.

10.2.2. Sobreutilización del concepto de Clase y de Instancia

Al no habernos basados en la orientación a objetos, no hemos necesitado utilizar en ningún momento el concepto de *class* que se utiliza para modelizar según las propuestas de OMG. Tampoco hemos tenido que definir diferentes interpretaciones para el término modelo según el nivel de abstracción. Así mismo, puesto que sólo hay una definición general de modelo, no se produce la replicación de conceptos. El resultado, comparado con la modelización basada en clases, es mucho más simple.

Por otro lado, nuestra teoría ofrece un mecanismo simple para determinar si una entidad es un original (el equivalente a *qué es una instancia* en la teoría OMG). En efecto cualquier entidad puede ser fácilmente identificada como original de un modelo mediante la función de validación del modelo. Igualmente, se definen qué mecanismos (los constructores) son los responsables de la creación del modelo a partir de una muestra de originales o para la creación de originales a partir del modelo (el equivalente a la instanciación). Finalmente, la teoría expuesta no requiere que toda entidad tenga que ser el original de un modelo. De este modo se suprime la necesidad constante de utilizar el concepto de original, tal como ocurre con el de instancia en el enfoque OMG.

10.2.3. Carencia de diagramas orientados a objetos

El problema planteado sobre la carencia de diagramas especializados en orientación a objetos que no separen estructura de comportamiento, se aborda desde nuestra teoría como una cuestión relacionada con las interrelaciones entre modelos y lenguajes, y más concretamente con los lenguajes de modelización.

Ya nos hemos referido a la existencia de un metamodelo de la orientación a objetos (ver apartado 8.4.3) OOMM, y cómo existen lenguajes cuyo modelo lingüístico es homomórfico con OOMM. Estos son los lenguajes que podemos denominar *Lenguajes de modelización orientados a objetos*, ya que en virtud de la Definición 8.5 se trata de lenguajes de modelización.

Estos lenguajes pueden ser gráficos o textuales. Incluso, podríamos usar OOMM sin expresarlo. En cualquier caso, si el lenguaje es verdaderamente orientado a objetos, el homomorfismo al que hemos hecho referencia debería ser un isomorfismo y de este modo requerir que su modelo lingüístico contenga sentencias sobre el lado estático y dinámico puesto que así ocurre en OOMM. Ésta es la razón por la que estos lenguajes son apropiados para modelizar la orientación a objetos.

Podríamos decir que UML, y más concretamente el Diagrama de clases de UML es un (sub)lenguaje orientado a objetos pero existen algunas diferencias importantes entre los lenguajes de modelización concebidos desde nuestra teoría y los existentes según OMG. En UML la construcción de modelos basados en clases siempre conduce a lo que hemos denominado *modelos sintéticos* debido a la naturaleza sintética del concepto de *class* definido por OMG. Nuestro punto de vista es más general puesto que permite la existencia de modelos puramente analíticos, y esta distinción debería ser contemplada en un lenguaje de modelización orientado a objetos para que se produjera el homomorfismo (más bien un isomorfismo) al que hemos hecho referencia. Desde nuestra perspectiva, UML se centra en el protomodelo de OOMM y deja de lado la expresión de sus constructores y por tanto con sus diagramas se pierde la posibilidad de transmitir el carácter analítico, analógico o sintético de un modelo.

La teoría expuesta no pretende ofrecer en sí misma una especificación de un lenguaje de modelización alternativo a UML pero sí expresa qué requisitos debe cumplir para que pueda serlo en realidad. Y estos requisitos pasan por expresar todos los elementos de cualquier modelo en general (protomodelo y constructores) y por ser fiel (isomórficamente) al metamodelo al que está asociado.

Otro aspecto en el que hemos incidido insistentemente es el de la importancia

de no confundir el metamodelo con el lenguaje, tal como ocurre con UML visto como lenguaje y como metamodelo. Un Diagrama de clases *no* es un modelo, aunque así se afirme de manera informal y no tan informal. Es una expresión incompleta (pues sólo refleja la parte estática) de un protomodelo orientado a objetos. Por tanto no es de extrañar que algunos autores (véase por ejemplo Harel and Rumpe (2000)) hayan encontrado serias dificultades para identificar el modelo semántico que hay detrás de un Diagrama de clases UML y nuestra teoría explica la razones de esas dificultades.

10.2.4. Complejidad de modelo frente a sistema modelizado

Finalmente veamos nuestro punto de vista sobre el problema de que un modelo pudiera ser más complejo que sus originales, como ocurre la mayoría de los estándares de modelización propuestos por OMG. Nuestra teoría deja claro que en el caso de modelos analíticos y analógicos eso no debe ser posible puesto que existe primero una reducción debida a la observación que hace que un original percibido sea más simple que un original real, y luego una abstracción (selección de las características a las que harán referencia las sentencias del modelo) mediante la cual también se produce una simplificación respecto al original percibido.

En el caso de los modelos sintéticos el proceso es muy diferente pero, tal como hemos explicado en la sección 6.7, también conduce a que el modelo es una simplificación respecto al *original concebido*, ya que éste se concibe precisamente a partir de una semillas definidas como aquellos elementos que permiten asemejarlo a un original concebido. Desde nuestro punto de vista la confusión proviene de no distinguir entre original abstracto y original concebido en un modelo sintético.

10.3. Dificultades sobre los niveles de la modelización

Veamos a continuación los problemas relacionados con los niveles de abstracción en la modelización. Ante todo debemos aclarar que nuestra teoría no pretende reeditar el enfoque de 4 o más niveles sino que por el contrario se centra exclusivamente en la relación entre modelo y original.

10.3.1. Necesidad de un modelo raíz mínimo

Aunque hemos presentado una definición de *metamodelo*, basada en una de las acepciones ofrecida por la literatura, no requerimos que todo modelo sera original de un metamodelo, como el diagrama de los 4 niveles de OMG sugiere (ver figura 4.16), ni requerimos la existencia de un modelo raíz ubicado en la cima de esa arquitectura de 4 capas. De hecho, existen muchos modelos creados a partir de un proceso analítico y usados a lo largo del tiempo, sobre los cuales en ningún momento se les ha definido un metamodelo del que fuesen originales.

Esto no impide, sin embargo, que podamos abordar el problema del modelo raíz desde la óptica de la teoría expuesta. Nuestra respuesta adopta un enfoque muy diferente definiendo un nuevo tipo de modelos (que no son lo mismo que metamodelos) a los que hemos denominado *modelos universales*. Desde esa óptica desaparecen muchos de los interrogantes alrededor de los modelos raíz. Nuestra conjetura es que quizás si nadie ha conseguido que un determinado modelo sea aceptado por la literatura como modelo raíz es probablemente porque tal modelo no existe realmente y los intentos de definirlo (como por ejemplo MOF) llevan indefectiblemente hacia la confusión, la replicación de conceptos y todos los problemas ya descritos en la sección 4.3.

La visión de los modelos universales es mucho más simple porque son como cualquier otro modelo, pero reflexivos, y modelos de todos los originales y metaoriginales en un dominio determinado. Aún más, proponemos el modelo de sistema como el ejemplo de modelo universal más claro y fácil de entender por cualquier humano, y que dista mucho de la complejidad y confusión asociada a las (meta)(meta)clases de MOF, por ejemplo.

10.3.2. Bidireccionalidad entre modelo y entidad modelizada

La teoría expuesta establece que sólo se puede hablar de modelos cuando existe al menos un constructor. En caso contrario estamos ante un protomodelo. Decimos que un modelo es bidireccional cuando tiene al mismo tiempo constructores analíticos/analógicos y sintéticos y en ese caso lo denominamos *modelo completo*. En este tipo de modelos, los cambios en los originales afectan al modelo (por su carácter analítico/analógico) y viceversa (por su carácter sintético).

10.3.3. Cardinalidad entre modelo y entidad modelizada

Nuestra teoría va mucho más allá al considerar no sólo las relaciones entre modelos y originales sino también entre diferentes tipos de originales: observables (generalmente físicos), percibidos/concebidos/análogos (como son vistos por el modelizador) y abstractos (una vista parcial del original percibido/concebido/análogo simplificada por el modelo). De estos tipos de originales, el único que merece la consideración de denominarlo *original* es el percibido/concebido/análogo puesto que es el único sobre el que el modelizador puede obtener una verdera abstracción seleccionando características relevantes. En un original abstracto, por ejemplo no se produce ninguna selección de características relevantes puesto que todos los valores de esas características ya son relevantes.

La confusión entre estos tipos de originales conduce a deducir (muchas veces erróneamente) diferentes cardinalidades respecto al modelo (ver apartado 6.7.4 sobre *modelos unitarios*). La teoría expuesta ofrece una solución coherente y entendible sobre este problema.

10.4. Dificultades sobre Metamodelización

10.4.1. Estereotipos UML

Toda la problemática sobre los estereotipos descrita en la sección 4.4.1 no tiene ningún sentido desde nuestro punto de vista, puesto que ha sido producto de una necesidad de cubrir algunas de las carencias de las propuestas de OMG. En nuestra teoría no existe el dilema sobre si usar estereotipos o MOF puesto que como hemos indicado el modelizador debe centrar su atención en definir correctamente sus modelos identificando su carácter analítico, analógico o sintético y un dominio de originales bien definido sobre los que crear sus modelos.

10.4.2. Herencia y metamodelización

La respuesta de nuestra teoría al problema de la confusión entre herencia y metamodelización es clara: la modelización no necesita el concepto de herencia y en su lugar hemos definido una relación de submodelización con una definición simple, clara y precisa, y deduciendo sus propiedades como hemos comentado más arriba. Lo mismo puede decirse de la metamodelización.

Gracias a esas definiciones, hemos podido deducir que efectivamente la submodelización es transitiva y la metamodelización es intransitiva (aunque no antitransitiva como sugiere Kühne (2006a)). Pero nuestra diferenciación de la metamodelización respecto a submodelización va más allá puesto que hemos comprobado que las principales diferencias radican en los protomodelos (que deben *hablar* sobre constructores en el caso de metamodelos, pero no necesariamente en el de los submodelos) y en los constructores (que tienen pecularidades específicas en el caso de los metamodelos).

Conclusiones y líneas de trabajo futuro

El objetivo inicial de esta tesis fue el estudio de soluciones de seguridad en el ámbito de SOA. Desde el principio fuimos conscientes de que el contexto de la investigación, la seguridad, era un tema complejo. Por ese motivo, el enfoque fue el de abordar toda esa complejidad mediante modelos. De hecho, ése es el camino seguido por cualquier investigador en seguridad. En nuestro caso, el camino ha pasado por diferentes etapas, cada una de ellas con resultados novedosos y que describimos a continuación a modo de conclusiones.

El primer paso en esa singladura fue la definición de modelos conceptuales centrados en la seguridad del código. El resultado de esa investigación es la propuesta de un nuevo modelo conceptual de seguridad SOA cuya singularidad principal es la de abarcar todos los elementos que intervienen en una interacción desde un punto de vista integral, incluyendo el código como el gran olvidado en estudios similares. En esa línea se continuó con la posibilidad de aplicar el mismo enfoque a los datos que intervienen en una comunicación SOA. Entonces se comprobó que este nuevo modelo conceptual centrado en los datos era equivalente con el del código. Finalmente, la evolución natural de ese trabajo fue la de integrar ambos modelos en un modelo conceptual de seguridad que los agrupase en objetos. Estos modelos conceptuales eran compatibles con el modelo conceptual general desarrollado por OASIS para SOA pero al que aportamos unas mejoras que cubrían algunas de las deficiencias encontradas como la no consideración del código y de la figura de los actores.

El segundo paso fue aproximarnos hacia un modelo de seguridad SOA que permitiera pasar del plano conceptual desarrollado en la etapa anterior hasta una arquitectura de seguridad basada en nuestro enfoque integral. Para conseguir ese objetivo analizamos una amplia literatura clasificada considerando tres

dimensiones: los modelos de seguridad en sí, los modelos de SOA y los modelos que relacionan la seguridad con SOA. Respecto a la primera dimensión, además de la revisión de los principales modelos de seguridad, realizamos una propuesta ampliando las dimensiones de seguridad comunes en la literatura (todas ellas basadas en el modelo CIA). Esta aportación era necesaria para poder abordar una arquitectura integral. En la segunda dimensión realizamos un estudio de las principales propuestas que pudieran sernos válidas para nuestro objetivo. Finalmente, en el último grupo centramos nuestra atención especialmente en el estudio de las soluciones de ingeniería de software dirigida por modelos (MDE) que tienen su origen en la iniciativa MDA propuesta por OMG. Nuestro interés en este enfoque radica en que promueve la realización de soluciones basadas en modelos que permiten generar soluciones software mediante la transformación de modelos. Esa visión prometía ser aplicable igualmente a la seguridad SOA si se consiguieran definir adecuadamente los modelos de seguridad y de SOA para mediante transformaciones MDE obtener modelos de seguridad integral SOA. En este punto, ya nos encontramos algunos problemas que obstaculizaban el avance de la investigación. La cuestión fue que esas soluciones MDE están basadas en un conjunto de estándares sobre modelización propuestos por OMG y cuyas especificaciones eran realmente complejas y confusas. Pero no sólo eso: los casos que encontramos en la literatura en los que los autores intentaban aplicar una filosofía MDE mostraban que efectivamente había bastantes cuestiones aún no resueltas. Sirva de ejemplo el dilema de elegir entre una modelización basada en estereotipos o en MOF, siendo ambos estándares promovidos por OMG, pero con el inconveniente de que no quedaba claro qué consecuencias se derivaban de una u otra elección. Entendimos por tanto, que para continuar con la investigación era necesario un estudio más profundo de estos estándares y de todas las cuestiones abiertas que pudieran afectarnos a la hora de aplicarlas a nuestro caso concreto.

Se abre en ese momento un aparente paréntesis en nuestro camino, necesario para la consecución con éxito de nuestro objetivo. En esta tercera etapa, investigamos una ingente literatura sobre modelización que nos llevaba de una cuestión abierta a otra, con un colectivo de autores con puntos de vista bastante dispares en algunos casos. La primera conclusión de nuestro estudio es que había muchos problemas de fondo en la modelización y comenzar a aplicar las propuestas de OMG eran como construir un edificio complejo con unos cimientos que no eran sólidos. El primer y principal problema de fondo era la propia nomenclatura.

Ni siquiera el concepto de modelo (obviamente fundamental en la modelización) estaba claro. Encontramos más de 20 definiciones diferentes, algunas de ellas parecidas pero con matices difíciles de analizar. Por ese motivo, decidimos buscar alguna herramienta de presentación y organización del conocimiento que nos ayudase en esa labor. El estudio, que también es una de las aportaciones de la tesis, es según nuestro conocimiento uno de los más completos que se han realizado hasta la fecha y que resumimos en una tabla comparativa detallada de las características más relevantes. De todas las herramientas analizadas, la que más se acercaba a solucionar nuestras necesidades era Concept Maps. Efectivamente nos servía para representar gráficamente conceptos y relaciones entre conceptos de cada definición, pero no permitía una comparación entre definiciones sin perder la autoría de cada una de ellas. Decidimos entonces ampliar y mejorar Concept Maps con dos mecanismos (path labeling y concept layering) que demostraron ser muy eficaces para lograr nuestro objetivo. El resultado fue la definición y publicación de una variante de Concept Mapas que denominamos RCM, tomando como primer ejemplo de uso el de modelo. A partir de ahí, desarrollamos un conjunto de métricas asociadas a los RCM que permiten al investigador obtener información relevante sobre los conceptos y su relación con las referencias. Tras realizar el RCM de modelo, continuamos con el de metamodelo, lenguaje y lenguaje de modelización. El estudio de los RCM nos dio valiosa información sobre los conceptos que subyacen en las definiciones y las relaciones entre ellos.

Tras el paréntesis que supuso el desarrollo y aplicación de RCM, continuamos con nuestro estudio de los problemas de modelización. El resultado de esta etapa es una relación de cuestiones abiertas clasificadas en tres grupos, cuyo resumen fue presentado en forma de *Guía de supervivencia* para un estudioso no experto de la modelización. Esta investigación cuyos resultados fueron publicados en el Congreso MODELS captó el interés de los asistentes y es otra de las aportaciones de la tesis. Todo ello confirmó nuestras sospechas de que los problemas encontrados no eran únicamente producto de nuestra inexperiencia con las soluciones de modelización basadas en OMG, sino que también, o más bien sobre todo, tenían su origen en las inconsistencias, ambigüedades, contradicciones, confusiones y complejidad de estas propuestas.

En este punto, llegamos a la conclusión de que no era posible solucionar esos problemas, ni siquiera sortearlos, puesto que entendimos que el enfoque en sí (la adopción de la orientación a objetos como base para la modelización) era

el desencadenante del resto de los problemas. Se necesitaba por tanto que el cambio de paradigma desde la orientación a objetos hacia la modelización se produjera como un salto hacia delante en lugar de una fusión forzada entre los dos paradigmas. A partir de ese momento la investigación se centra en la búsqueda de una propuesta de teoría sobre los fundamentos de la modelización *no contaminada* y en la que el centro sea realmente el modelo y no el objeto (o la clase de objetos). La solución obtenida se basa en tres principios: el método científico, el principio de independencia (basado en el triángulo del significado) y la visión general de la modelización. La aplicación de estos tres principios es una de las principales aportaciones de la tesis puesto que nos conducen a una visión diferente de la modelización que da lugar a una teoría consistente y con unos principios sólidos y más fáciles de entender tanto para el inexperto como el experto en modelización. A modo de resumen, algunos de los principales resultados fueron los siguientes:

- Caracterización de los principales tipos de modelos según un enfoque científico.
- Introducción de los conceptos de modelo analógico, modelo completo y modelo unitario.
- Diferenciación entre modelo y protomodelo, a través de la introducción de los constructores del modelo.
- Aclaración de la problemática de lo físico y abstracto mediante una teoría clara y consistente.
- Separación de los conceptos de lenguaje y sistema en la definición de modelo.
- Estudio de las principales relaciones entre modelos: submodelización, metamodelización, caracterización y transformación, con interesantes conclusiones que ofrecen una respuesta razonada a muchas de las cuestiones planteadas sobre los estándares de modelización de OMG.
- Explicación de la metamodelización según el modelo científico. Descubrimiento de interesantes relaciones entre los constructores de un modelo y su metamodelo.
- Enunciado de las propiedades que tienen esas relaciones entre modelos.
- Definición de modelos universales y propuesta del modelo de sistema como modelo universal y por tanto reflexivo y mínimo.
- Definición de lenguaje basada en la de modelo a través de la introducción del metamodelo y modelo lingüístico.

- Definición de lenguaje de modelización y metalenguaje desde un punto de vista que clarifica muchas de las cuestiones y ambigüedades debatidas en la literatura.
- Estudio de la pragmática de la modelización basándose en el resto de la teoría y obteniendo interesantes conclusiones sobre el proceso de comunicación de modelos.

El proceso de elaboración de la teoría ha sido complejo y laborioso debido a la multitud de matices a los que había que dar respuesta y la enorme cantidad de referencias que ha sido necesario consultar. El resultado final ha sido puesto a prueba dedicando un capítulo a retomar los principales problemas de la modelización que sirvieron como motivación para el desarrollo. En ese capítulo se realiza una descripción de las respuestas que ofrece esta teoría para aclarar o explicar todos esos problemas. Se ha demostrado que es posible definir una teoría sobre los fundamentos de la modelización no basada en la orientación a objetos y centrada en dar respuesta a todos los interrogantes que han dado lugar a tantos debates en la literatura sobre modelización.

Lineas de trabajo futuro

Lógicamente de las etapas descritas se deduce que la principal línea de trabajo futuro es la aplicación práctica de nuestra propuesta para la resolución de diferentes problemas de modelización, como por ejemplo abordar la construcción de un metamodelo de seguridad partiendo de nuestro trabajo inicial sobre modelos conceptuales de seguridad integral SOA. No obstante, como reza el principio de generalidad sobre el que se basa la teoría, la vocación de este trabajo es que pueda ser aplicado no sólo a la modelización de la seguridad sino en cualquier otro ámbito de estudio. En ese sentido, las líneas de actuación serían principalmente las siguientes:

- Implementación sobre sistemas software de constructores de modelos (analíticos, analógicos y sintéticos) para la generación de protomodelos y de originales según la teoría propuesta.
- Desarrollo de sistemas de información que permitan la ejecución de los roles descritos en el capítulo 9 sobre la pragmática de la modelización.
- Aplicación de los fundamentos sobre la transformación de modelos descritos

en la sección 7.4. El estudio de las transformaciones de modelos desde la óptica de nuestra teoría permitiría aplicar el enfoque MDE pero desde la perspectiva de unos fundamentos de la modelización consistentes y con una base conceptual más simple que la ofrecida actualmente.

Respecto a los RCM, las principales líneas de actuación futuras serían las siguientes:

- Profundización en el desarrollo de nuevas herramientas que permitan la realización en línea de RCM auténticamente colaborativos
- Aplicación de los RCM en otros ámbitos diferentes al del estudio de la terminología.

Finalmente, planteamos como trabajo futuro una comparativa exhaustiva con la Fenomática de Dominguez (1999), así como desarrollar con mayor profundidad las implicaciones que tiene la definición del modelo de sistema como un modelo universal.

Bibliografía

M. AAhlberg. Varieties of concept mapping. In *Proc. of the First Int. Conference on Concept Mapping*, pages 14–17, Pamplona (Spain), 2004.

DH Akehurst, WGJ Howells, B. Bordbar, and KD Mcdonald-Maier. Maths vs (Meta) Modelling: Are we reinventing the Wheel? In *Third International Conference on Software and Data Technologies (ICSOFT 2008)*, Porto (Portugal), 2008.

Masoom Alam, Ruth Breu, and Michael Hafner. Model-Driven Security Engineering for Trust Management in SECTET. *Journal of Software*, 2(1):47–59, 2007.

Rafik Amir and Amir Zeid. A UML profile for service oriented architectures. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, OOPSLA '04, pages 192–193, New York, 2004. ACM. ISBN 1581138334.

Ali Arsanjani. Service-oriented modeling and architecture, 2004. URL http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/.

Uwe Aß mann, Steffen Zschaler, and Gerd Wagner. Ontologies, Meta-models, and the Model-Driven Paradigm. In *Ontologies for Software Engineering and Software Technology*, pages 249–273. Springer Berlin Heidelberg, 2006. doi: 10.1007/3-540-34518-3_9.

C Atkinson, Andrea de Maio, and R Bayan. Dragoon: an object-oriented notation supporting the reuse and distribution of Ada software. *Ada Letters*, X(9):50–59, 1990. ISSN 1094-3641. doi: 10.1145/102456.102470.

Colin Atkinson and Thomas Kühne. The Essence of Multilevel Metamodeling. In Cris Gogolla, Martin and Kobryn, editor, *UML 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, volume 2185 of *Lecture Notes in Computer Science*, pages 19–33, Toronto, 2001. Springer Berlin / Heidelberg. ISBN 978-3-540-42667-7.

Colin Atkinson and Thomas Kühne. Rearchitecting the UML infrastructure. *ACM Trans. Model. Comput. Simul.*, 12(4):290–321, October 2002a. ISSN 1049-3301. doi: 10.1145/643120.643123.

Colin Atkinson and Thomas Kühne. Profiles in a strict metamodeling framework. *Science of Computer Programming*, 44(1):5–22, July 2002b.

Colin Atkinson and Thomas Kühne. Model-driven development: a metamodeling foundation. *IEEE Software*, 20(5):36–41, September 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231149.

Colin Atkinson and Thomas Kühne. Concepts for Comparing Modeling Tool Architectures. In Lionel Briand and Clay Williams, editors, *Proceedings of the ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, MoDELS / UML*, volume 3713 of *Lecture Notes in Computer Science*, pages 398–413. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-29010-0. doi: 10.1007/11557432_30.

Colin Atkinson, Brian Henderson-Sellers, and Thomas Kühne. To Meta or Not to Meta. That Is the Question. *Journal of Object-Oriented Programming*, 13 (8):32–35, December 2000.

Colin Atkinson, Thomas Kühne, and B. Henderson Sellers. Systematic stereotype usage. *Software and Systems Modeling*, 2(3):153–163, 2003.

D.P. Ausubel. *The psychology of meaningful verbal learning.* Grune & Stratton, New York, 1963.

Thomas Baar. Correctly defined concrete syntax for visual models. In Gianna Nierstrasz, Oscar and Whittle, Jon and Harel, David and Reggio, editor, *Model Driven Engineering Languages and Systems*, volume 4199, pages 111–125. Springer Berlin Heidelberg, 2006. doi: 10.1007/11880240_9.

Harriet E. Baber. Language: Saying the Same Thing. In *A Prelude to Logic*, chapter 2. Lulu Press, San Diego, 2011. ISBN 9781257712380.

Aymen Baouab, Olivier Perrin, Nicolas Biri, and Claude Godart. Security Meta-Services Orchestration Architecture. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 28 –33, 2009.

Mikaël Barbero, Frédéric Jouault, Jeff Gray, and Jean Bézivin. A Practical Approach to Model Extension. In *MDA-Foundations and Applications*, pages 32–42. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-72901-3_3. URL http://dx.doi.org/10.1007/978-3-540-72901-3_3.

D. Basin, M. Clavel, and M. Egea. A decade of model-driven security. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 1–10, 2011.

David Basin, J\ürgen Doser, and Torsten Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, January 2006. doi: 10.1145/1125808.1125810.

Gorka Benguria, Xabier Larrucea, Brian Elvesæ ter, Tor Neple, Anthony Beardsmore, and Michael Friess. A Platform Independent Model for Service Oriented Architectures. In Guy Doumeingts, Jörg Müller, Gérard Morel, and Bruno Vallespir, editors, *Enterprise Interoperability*, pages 23–32. Springer London, 2007. ISBN 978-1-84628-714-5.

Jean Bézivin. Towards a precise definition of the OMG/MDA framework. In *Proceedings. 16th Annual International Conference on Automated Software Engineering, 2001.(ASE 2001)*, pages 273–280. IEEE Computer Society, 2001.

Jean Bézivin. On the unification power of models. *Software and Systems Modeling*, 4(2):171–188, May 2005. ISSN 1619-1366. doi: 10.1007/s10270-005-0079-0.

Jean Bézivin and Richard Lemesle. Ontology-Based Layered Semantics for Precise OA&D Modeling. In *Proceedings of the Workshops on Object-Oriented Technology*, ECOOP '97, pages 151–154, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-64039-8.

Jean Bézivin, Frédéric Jouault, Peter Rosenthal, and Patrick Valduriez. Modeling in the Large and Modeling in the Small. In *Model Driven Architecture*, pages 33–46. Springer Berlin Heidelberg, 2005. doi: 10.1007/11538097_3.

Jean Bézivin, Fabian Büttner, Martin Gogolla, Frederic Jouault, Ivan Kurtev, and Arne Lindow. *Model Transformations? Transformation Models!* Springer-Verlag, Genova, Italy, 2006. doi: 10.1007/11880240_31.

M. Biezunski, M. Bryan, and S. Newcomb. ISO/IEC 13250 Topic Maps. Journal article, International Organization for Standardization, 2002.

Paul Biron, Ashok Malhotra, World Wide Web Consortium, and Others. XML schema part 2: Datatypes. *World Wide Web Consortium Recommendation*, 2004. URL http://www.w3.org/TR/xmlschema-2/.

Paul Bloom and Frank C. Keil. Thinking Through Language. *Mind & Language*, 16(4):351–367, 2001. ISSN 1468-0017. doi: 10.1111/1468-0017.00175.

D. Booth, H. Haas, and A. Brown. Web Services Glossary, February 2004a. URL http://www.w3.org/TR/ws-gloss/.

David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture, W3C Working Group Note, February 2004b. URL http://www.w3.org/TR/ws-arch/.

A. Boronat, I.R.U.P. Valencia, S.D.J.A. Carsi-UP, and S. Valencia. *MOMENT: a formal framework for MOdel manageMENT*. Phd in computer science, Universitat Politenica de Valencia (UPV), Spain, 2007.

Jürgen Börstler, Ludwik Kuzniarz, Carl Alphonce, William B Sanders, and Michal Smialek. Teaching software modeling in computing curricula. In *Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups*, ITiCSE-WGR '12, pages 39–50, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1872-3. doi: 10.1145/2426636.2426640.

Paolo Bottoni, Fulvio D'Antonio, and Michele Missikoff. Towards a Unified View of Model Mapping and Transformation. In Michele Missikoff, Antonio De Nicola, and Fulvio D'Antonio, editors, *Proceedings of the Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability (EMOI-INTEROP),*

co-located with CAiSE 2006 Conf., volume 200 of *CEUR Workshop Proceedings*. CEUR-WS.org, June 2006.

Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan {&} Claypool, 2012. ISBN 9781608458820.

Ruth Breu, Michael Hafner, Frank Innerhofer-Oberperfler, and Florian Wozak. Model-Driven Security Engineering of Service Oriented Systems. In Clemens Kaschek, Roland and Kop, Christian and Steinberger, Claudia and Fliedl, Günther and Aalst, Wil and Mylopoulos, John and Rosemann, Michael and Shaw, Michael J. and Szyperski, editor, *Information Systems and e-Business Technologies*, volume 5 of *Lecture Notes in Business Information Processing*, pages 59–71. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-78942-0.

Sjaak Brinkkemper. Method engineering: engineering of information systems development methods and tools. *Information and Software Technology*, 38(4): 275–280, 1996. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/0950-5849(95)01059-9.

Roberto Bruni, Matthias Hölzl, Nora Koch, Alberto Lluch Lafuente, Philip Mayer, Ugo Montanari, Andreas Schroeder, and Martin Wirsing. A service-oriented UML profile with formal support. In Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors, *Service-Oriented Computing*, volume 5900 of *Lecture Notes in Computer Science*, book part (with own title) 34, pages 455–469. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-10382-7.

R. Bunge, S. Chung, B. Endicott-Popovsky, and D. McLane. An Operational Framework for Service Oriented Architecture Network Security. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 312–312, 2008. doi: 10.1109/HICSS.2008.64.

F. Buttner and M. Gogolla. On generalization and overriding in UML 2.0. In *OCL and Model Driven Engineering, UML 2004 Conf. Workshop, O. Patrascoiu, Ed. Univ. Kent*, pages 1—-15, 2004.

T. Buzan. *Use your head*. Pearson Education, London, UK, 1974.

A.J. Cañas and M. Carvalho. Concept Maps and AI: an Unlikely Marriage? In *Proceedings of SBIE 2004: Simpósio Brasileiro de Informática Educativa*, volume 1, pages 1–10, Manaus, Brasil, 2004.

Alberto Cañas, Roger Carff, Greg Hill, Marco Carvalho, Marco Arguedas, Thomas Eskridge, James Lott, and Rodrigo Carvajal. Concept Maps: Integrating Knowledge and Information Visualization. In Sigmar-Olaf Tergan and Tanja Keller, editors, *Knowledge and Information Visualization*, volume 3426 of *Lecture Notes in Computer Science*, pages 181–184. Springer Berlin Heidelberg, 2005. doi: 10.1007/11510154_11.

M.J. Carnot, P. Feltovich, R.R. Hoffman, J. Feltovich, and J.D. Novak. A summary of literature pertaining to the use of concept mapping techniques and technologies for education and performance support. Technical report, Technical Report submitted to The Chief of Naval Education and Training Pensacola FL, 2003.

Kai Chen, Janos Sztipanovits, and Sandeep Neema. Toward a semantic anchoring infrastructure for domain-specific modeling languages. In *Proceedings of the 5th ACM international conference on Embedded software*, EMSOFT '05, pages 35–43, Jersey City, NJ, USA, 2005. ACM. ISBN 1-59593-091-4. doi: 10.1145/1086228.1086236.

Peter Pin-Shan Chen. The entity-relationship model. Toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976. ISSN 0362-5915. doi: 10.1145/320434.320440.

Gennaro Chierchia and Raymond Turner. Semantics and property theory. *Linguistics and Philosophy*, 11(3):261–302, August 1988. ISSN 0165-0157. doi: 10.1007/BF00632905.

N. Chomsky. *Syntactic structures*, volume 2. Walter de Gruyter, Berlin, 1957.

P. Ciccarese, M. Ocana, L.J.G. Castro, S. Das, and T. Clark. An open annotation ontology for science on web 3.0. *Journal of biomedical semantics*, 2(Suppl 2): S4, 2010.

Manuel Clavel, Viviane da Silva, Christiano Braga, and Marina Egea. Model-Driven Security in Practice: An Industrial Experience. In Ina Schieferdecker and Alan Hartman, editors, *Model Driven Architecture – Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 326–337. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-69095-5.

Martin Davies. Concept mapping, mind mapping and argument mapping: what are the differences and do they matter? *Higher Education*, 62(3):1–23, November 2010. ISSN 0018-1560. doi: 10.1007/s10734-010-9387-6.

Valeria De Castro, Esperanza Marcos, and Juan Manuel Vara. Applying CIM-to-PIM model transformations for the service-oriented development of information systems. *Information and Software Technology*, 53(1):87–105, January 2011. ISSN 09505849.

Ferdinand De Saussure. *Course in General Linguistics*, volume 240. Columbia University Press, 1974.

J. Dingel, Z. Diskin, and A. Zito. Understanding and improving UML package merge. *Software & Systems Modeling*, 7(4):443–467, October 2008. ISSN 1619-1366. doi: 10.1007/s10270-007-0073-9.

E Dominguez and M A Zapata. Noesis: Towards a situational method engineering technique. *Information Systems*, 32(2):181–222, 2007. ISSN 0306-4379. doi: http://dx.doi.org/10.1016/j.is.2005.07.001.

Eladio Dominguez. Un paseo fenomático. Technical report, Academia de Ciencias Exactas, Físicas, Químicas y Naturales de Zaragoza, 1999.

J Duntemann and C Marinacci. New Objects for Old Structures: Using object-oriented techniques to convert existing applications has its advantages. *Byte*, 15(4):261–266, 1990. ISSN 0360-5280.

C. Eden. Cognitive mapping. *European Journal of Operational Research*, 36(1): 1–13, 1988.

Hany F. EL Yamany, Miriam A.M. Capretz, and David S. Allison. Intelligent security and access control framework for service-oriented architecture. *Inf. Softw. Technol.*, 52(2):220–236, February 2010. ISSN 0950-5849. doi: 10.1016/j.infsof.2009.10.005.

Niklas Elmqvist and Philippas Tsigas. CiteWiz: a tool for the visualization of scientific citation networks. *Information Visualization*, 6(3):215–232, 2007. ISSN 1473-8716. doi: 10.1145/1375939.1375943.

M.J. Eppler. A comparison between concept maps, mind maps, conceptual diagrams, and visual metaphors as complementary tools for knowledge construction and sharing. *Information Visualization*, 5(3):202–210, 2006.

Jean-Marie Favre. Foundations of Model (driven) (Reverse) Engineering - Episode I: Story of the Fidus Papyrus and the Solarus. In *Dagsthul Seminar on Model Driven Reverse Engineering*, 2004a.

Jean-Marie Favre. Towards a Basic Theory to Model Model Driven Engineering. In *3rd Workshop in Software Model Engineering, WiSME*, October 2004b.

Jean-Marie Favre. Foundations of meta-pyramids: Languages vs. metamodels Episode II: Story of thotus the baboon. In *Language Engineering for Model-Driven Software Development, number 04101 in Dagstuhl Seminar Proceedings*, volume 4101, 2005.

Jean-Marie Favre. Megamodelling and Etymology. In James R. Cordy, Ralf Lämmel, and Andreas Winter, editors, *Transformation Techniques in Software Engineering. Dagstuhl Seminar Proceedings*, volume 5161 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

Jean-Marie Favre and Tam Nguyen. Towards a megamodel to model software evolution through transformations. *Electronic Notes in Theoretical Computer Science*, 127:59–74, 2004.

Martin Feilkas. How to represent Models, Languages and Transformations. In *Proceedings of the 6th OOPSLA workshop on domain-specific modeling (DSM'06)*, pages 169—-176, 2006.

J. Fiere. *SOA Security*. Thesis (phd), Vrije Universiteit Amsterdam, 2007.

Rony G. Flatscher. Metamodeling in EIA/CDIF meta-metamodel and metamodels. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 12(4):322–342, 2002. ISSN 1049-3301. doi: 10.1145/643120.643124.

Robert B France, Sudipto Ghosh, Trung Dinh Trong, and Arnor Solberg. Model-Driven Development Using UML2.0: Promises and Pitfalls. *Computer*, 39(2): 59–66, February 2006. ISSN 0018-9162. doi: 10.1109/MC.2006.65.

David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, Indianapolis, Indiana, January 2003. ISBN 0471319201.

Michael Franz, Deepak Chandra, Andreas Gal, Vivek Haldar, Fermin Reig, and Ning Wang. A portable Virtual Machine target for Proof-Carrying Code. In *Proceedings of the 2003 workshop on Interpreters, virtual machines and emulators*, volume 57, pages 24—-31. ACM Press, 2003. ISBN 1581136552. doi: 10.1145/858570.858573.

Robert M. French. The computational modeling of analogy-making. *Trends in Cognitive Sciences*, 6(5):200–205, May 2002. ISSN 13646613. doi: 10.1016/S1364-6613(02)01882-X.

Jose M Fuentes, Victor Quintana, Juan Llorens, Gonzalo Genova, and Rubén Prieto Diaz. Errors in the UML metamodel? *SIGSOFT Software Engineering Notes*, 28(6):3–3, November 2003. ISSN 0163-5948. doi: 10.1145/966221.966236.

H.D.D. Garcia. *Evaluation of Data-Centric Process Modeling Approaches*. Thesis (master's), Eindhoven University of Technology, Eindhoven, August 2011.

Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. A semantic framework for metamodel-based languages. *Automated Software Engineering*, 16 (3):415–454, December 2009. doi: 10.1007/s10515-009-0053-0.

Dragan Gašević, Nima Kaviani, and Marek Hatala. On Metamodeling in Megamodels. In Gregor Engels, Bill Opdyke, Douglas C. Schmidt, and Frank Weil, editors, *Model Driven Engineering Languages and Systems*, pages 91–105. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-75209-7_7.

R. Gitzel and T. Hildenbrand. A Taxonomy of metamodel hierarchies. Technical report, Department of Information Systems. University of Mannheim, 2005.

R. Gitzel and A. Korthaus. The Role of Metamodeling in Model-Driven Development. In *Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI2004)*, pages 68–73, 2004.

Ralf Gitzel, Ingo Ott, and Martin Schader. Ontological Extension to the MOF Metamodel as a Basis for Code Generation. *The Computer Journal*, 50(1): 93–115, January 2007. doi: 10.1093/comjnl/bxl052.

Lila Gleitman and Anna Papafragou. Language and thought. In Keith J Holyoak and Robert G Morrison, editors, *Cambridge handbook of thinking and reasoning*, book chapter/section 26, pages 633–661. Cambridge University Press Cambridge, UK, 2005.

Robert L Goldstone and Ji Yun Son. Similarity. *Psychological Review*, 100: 254–278, 2004.

Cesar Gonzalez-Perez and Brian Henderson-Sellers. Modelling software development methodologies: A conceptual foundation. *Journal of Systems and Software*, 80(11):1778–1796, November 2007. ISSN 01641212. doi: 10.1016/j.jss.2007.02.048.

Luis Borges Gouveia. A Brief Survey on Cognitive Maps as Humane Representations. *Mind*, 4:450—-469, March 2004.

Barry Gower. *Scientific Method: A Historical and Philosophical Introduction*. Routledge, New York, 2002. ISBN 0203046129.

Herbert Granger. Aristotle on Genus and Differentia. *Journal of the History of Philosophy*, 22:1–23, 1984.

Yann-gaël Guéhéneuc, Hervé Albin-amiot, Rémi Douence, and Pierre Cointe. Bridging the gap between modeling and programming languages. Electronic citation, Computer Science Department, école des Mines de Nantes, Nantes, 2002.

G Guizzardi. On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta) Models. In *Databases and Information Systems IV: Selected Papers from the Seventh International Conference DBandIS'2006*, page 18. Ios Pr Inc, 2007.

Anil Gupta. Definitions (The Stanford encyclopedia of philosophy (Fall 2012 edition), 2012. URL http://plato.stanford.edu/archives/fall2012/entries/definitions/.

P. Gupta and J. Sykes. *The conceptual modeling process and the notion of a concept*. IGI Global, 2001.

Carlos Gutierrez, Eduardo Fernandez-Medina, and Mario Piattini. Web services enterprise security architecture: a case study. In *Proceedings of the 2005 workshop on Secure web services*, SWS '05, pages 10–19, Fairfax, VA, 2005. ACM Press. ISBN 1595932348.

Michael Hafner and Ruth Breu. Basic Concepts of SOA Security. In *Security Engineering for Service-Oriented Architectures*, book part (with own title) 3, pages 27–45. Springer Berlag, 1st edition, November 2010. ISBN 978-3-540-79538-4.

Michael Hafner, Ruth Breu, Berthold Agreiter, and Andrea Nowak. SECTET - An Extensible Framework for the Realization of Secure Inter-Organizational Workflows. *Journal of Internet Research*, 16(5):491–506, 2006.

C. Hahn, P. Dmytro, and K. Fischer. A model-driven approach to close the gap between business requirements and agent-based execution. In *Proceedings of the 4th Workshop on Agent-based Technologies and applications for enterprise interoperability (ATOP 2010) held in conjunction with AAMAS 2010*, pages 13–24, Toronto, 2010.

Vivek Haldar, Christian H. Stork, and Michael Franz. The source is the proof. In *Proceedings of the 2002 workshop on New security paradigms*, pages 69–73, New York, 2002. ACM Press. ISBN 158113598X. doi: 10.1145/844102.844114.

D. Harel and B. Rumpe. Modeling Languages: Syntax, Semantics and All That Stuff, Part I:The Basic Stuff. Technical report, Weizmann Institute of Science, 2000.

David Harel and Bernhard Rumpe. Meaningful Modeling: What's the Semantics of "Semantics¿ *Computer*, 37(10):64–72, 2004. ISSN 0018-9162. doi: 10.1109/MC.2004.172.

Pat Hayes, Thomas C. Eskridge, Raul Saavedra, Thomas Reichherzer, Mala Mehrotra, and Dmitri Bobrovnikoff. Collaborative knowledge capture in ontologies. In *Proceedings of the 3rd international conference on Knowledge capture*, K-CAP '05, pages 99–106, New York, 2005. ACM. ISBN 1-59593-163-5. doi: 10.1145/1088622.1088641.

Reiko Heckel, Marc Lohmann, and Sebastian Thöne. Towards a UML profile for service-oriented architectures. In *Workshop on Model Driven Architecture: Foundations and Applications (MDAFA'03)*, Enschede, 2003. University of Twente.

Brian Henderson-Sellers and Cesar Gonzalez-Perez. Uses and Abuses of the Stereotype Mechanism in UML 1.x and 2.0. *Model Driven Engineering Languages and Systems*, 4199:16–26, 2006. doi: 10.1007/11880240_2.

M. Hesse. Models and analogies. In *A companion to the philosophy of science*, pages 299—-397. University of Notre Dame Press, 2001.

Wolfgang Hesse. More matters on (meta-)modelling: remarks on Thomas Kühne's "matters". *Software and Systems Modeling*, 5(4):387–394, December 2006. ISSN 1619-1366. doi: 10.1007/s10270-006-0033-9.

Douglas R Hofstadter. Analogy as the core of cognition. *The analogical mind: Perspectives from cognitive science*, pages 499–538, 2001.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison Wesley, 3 edition, July 2006. ISBN 0321462254. URL htt.

P.J. Hurley. *A concise introduction to logic*. Wadsworth Pub Co, Boston, 2011.

John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 471–480, Waikiki, Honolulu, HI, USA, 2011. ACM. ISBN 978-1-4503-0445-0.

ISO. ISO/IEC 27000 Information technology — Security techniques — Information security management systems — Overview and vocabulary, May 2009.

Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, April 2004.

S. Jeary, A. Fouad, and K.T. Phalp. Extending the Model Driven Architecture with a preCIM level. In *MDABIZ 2008: 1st International Workshop on Business Support for MDA co-located with TOOLS EUROPE 2008*, Zurich, 2008. CEUR-WS.

Harshavardhan Jegadeesan and Sundar Balasubramaniam. An MOF2-based Services Metamodel. *Journal of Object Technology*, 7(8):71—-96, December 2008.

Simon Johnston. Modeling security concerns in service-oriented architectures, June 2004. URL http://www.michael-richardson.com/rup_classic/soa.rup_soma/guidances/whitepapers/resources/soa_modeling_security.pdf.

Simon Johnston. UML 2.0 Profile for Software Services, April 2005.

V. Jonnaganti. An Integrated Security Model for the Management of SOA Improving the attractiveness of SOA Environments through a strong Architectural Integrity. Journal article, University of Gothenburg, 2009.

Frédéric Jouault and Jean Bézivin. KM3:A DSL for Metamodel Specification. *Formal Methods for Open Object-Based Distributed Systems*, 4037:171–185, 2006.

Frédéric Jouault, Jean Bézivin, and Mikaël Barbero. Towards an advanced model-driven engineering toolbox. *Innovations in Systems and Software Engineering*, 5(1):5–12, 2009. ISSN 1614-5046. doi: 10.1007/s11334-009-0082-7.

Jelena Jovanovic and Dragan Gasevic. IntelLEO Annotations Ontology, April 2011. URL http://www.intelleo.eu/ontologies/annotations/spec/.

G.H. Joyce. *Principles of logic*. Longmans, Green and Co., London, 1916.

Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. In Stephen Jézéquel, Jean-Marc and Hussmann, Heinrich and Cook, editor, *UML 2002 — The Unified Modeling Language*, volume 2460 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-44254-7.

Martin Karlsch. *A model-driven framework for domain specific languages*. Journal article, Hasso-Plattner-Institute of Software Systems Engineering, Potsdam, 2007.

M. Karow, A. Gehlert, J. Becker, and W. Esswein. On the Transition from Computation Independent to Platform Independent Models. In *AMCIS Conference Proceedings*, pages 3913–3921, 2006.

K. Kasal, J. Heurix, and T. Neubauer. Model-Driven Development Meets Security: An Evaluation of Current Approaches. In *Proceedings of the 2011 44th Hawaii International Conference on System Sciences*, HICSS '11, pages 1–9. IEEE Computer Society, 2011. ISBN 978-0-7695-4282-9.

V. Kaugers and U. Sukovskis. Model-Driven Secure System Development Framework. *Database and Information Systems: Doctoral Consortium. University of Latvia*, 757:43–52, 2010.

Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, Inc, Hoboken, New Jersey, March 2008. ISBN 0470036664.

George J. Klir. *Facets of Systems Science*. Kluwer Academic/Plenum Publishers, New York, August 2001. ISBN 0306466236.

Jeff Kramer. Is abstraction the key to computing? *Commun. ACM*, 50(4):36–42, April 2007. ISSN 0001-0782. doi: 10.1145/1232743.1232745.

Thomas Kühne. What is a Model? In *Language Engineering for Model-Driven Software Development in Dagstuhl Seminar*, 2005.

Thomas Kühne. Matters of (Meta-) Modeling. *Software and Systems Modeling*, 5(4):395–401, December 2006a. ISSN 1619-1366. doi: 10.1007/s10270-006-0034-8.

Thomas Kühne. Clarifying matters of (meta-) modeling: an author's reply. *Software and Systems Modeling*, 5(4):395–401, December 2006b. ISSN 1619-1366. doi: 10.1007/s10270-006-0034-8.

Thomas Kühne. Contrasting Classification with Generalisation. In *Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling*, volume 96 of *APCCM '09*, pages 71—-78, Wellington, 2009. Australian Computer Society, Inc. ISBN 978-1-920682-77-4.

Thomas Kühne. An Observer-Based Notion of Model Inheritance. In Dorina Petriu, Nicolas Rouquette, and Ø ystein Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *Lecture Notes in Computer Science*, book part (with own title) 3, pages 31–45. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-16144-5.

I. Kurtev. Metamodels: Definitions of Structures or Ontological Commitments. In *Workshop on TOWERS of models. Collocated with TOOLS Europe 2007*, volume 2007, 2007.

Sabine Kuske, Martin Gogolla, Hans-Jörg Kreowski, and Paul Ziemann. Towards an integrated graph-based semantics for UML. *Software and Systems Modeling*, 8:403–422, 2009.

U. Lang and R. Schreiner. Model driven security management: Making security management manageable in complex distributed systems. In *Modeling Security Workshop in Association with MODELS 2008*, volume 8, Toulouse, 2008.

P.A. Le. *ConceptBib: A tool for integrating Concept Map and Bibliographic Management*. Thesis (phd), Monash University, 2004.

Jorge Lloret. *Un modelo para el diseño conceptual de base de datos: el modelo EAN*. PhD thesis, Universidad de Zaragoza, 1998.

T. Lodderstedt. *Model driven security from UML models to access control architectures*. Thesis (phd), Universitätsbibliothek Freiburg, 2004.

Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook, editors, *UML 2002 - The Unified Modeling Language : 5th International Conference, Dresden, Germany, September 30 - October 4, 2002. Proceedings*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441. Springer-Verlag, September 2002. ISBN 978-3-540-44254-7. doi: 10.1007/3-540-45800-X_33.

Marcos Lopez-Sanz and Esperanza Marcos. ArchiMeDeS: A model-driven framework for the specification of service-oriented architectures. *Inf. Syst.*, 37(3): 257–268, 2012.

Marcos López-Sanz, César Acuña, Carlos Cuesta, and Esperanza Marcos. UML Profile for the Platform Independent Modelling of Service-Oriented Architectures. In Flavio Oquendo, editor, *Software Architecture*, volume 4758 of *Lecture Notes in Computer Science*, book part (with own title) 30, pages 304–307. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-75131-1.

Tom Love. *Object Lessons: Lessons in Object-Oriented Development Projects*. Cambridge University Press, 1993. ISBN 0962747734.

Jochen Ludewig. Models in software engineering - an introduction. *Software and Systems Modeling*, 2(1):5–14, March 2003. doi: 10.1007/s10270-003-0020-3.

Alexander R. Luria. *Language and Cognition*. John Wiley & Sons Inc, 1979. ISBN 0471093025.

J. Lyons. *Language and linguistics*. Cambridge University Press, 2002.

CM MacKenzie, K. Laskey, F. McCabe, PF Brown, and R. Metz. Reference Model for Service Oriented Architecture V 1.0. Technical report, OASIS, 2006.

C. Marchand, JF D Ivernois, JP Assal, G. Slama, and R. Hivon. An analysis, using concept mapping, of diabetic patients' knowledge, before and after patient education. *Medical Teacher*, 24(1):90–99, 2002.

Francisco Martínez and Ambrosio Álvarez. A Precise Approach for the Analysis of the UML Models Consistency. In Jacky Akoka, Stephen Liddle, Il-Yeol Song, Michela Bertolotto, Isabelle Comyn-Wattiau, Willem-Jan van den Heuvel, Manuel Kolp, Juan Trujillo, Christian Kop, and Heinrich Mayr, editors, *Perspectives in Conceptual Modeling*, volume 3770 of *Lecture Notes in Computer Science*, pages 74–84. Springer Berlin / Heidelberg, 2005.

Y. Martinez, C. Cachero, and S. Melia Beigbeder. Evidencia empírica sobre mejoras en productividad y calidad en enfoques MDD: un mapeo sistemático. *REICIS: Revista Española de Innovacion, Calidad e Ingenieria del Software*, 7 (2):6–27, 2011.

R. McAleese. The knowledge arena as an extension to the concept map: Reflection in action. *Interactive Learning Environments*, 6(3):251–272, 1998.

J.R. McCumber. National Training Standard for Information Systems Security (Info-Sec) Professionals - Reference Number 4011, June 1994.

F. Mehr and U. Schreier. Modelling of Message Security Concerns with UML. In *9th ICEIS*, 2007.

S.J. Mellor, K. Scott, A. Uhl, D. Weise, and R.M. Soley. *MDA distilled: principles of model-driven architecture*, volume 88. Addison-Wesley, New York, 2004.

M. Memon, M. Hafner, and R. Breu. SECTISSIMO: A Platform-independent Framework for Security Services. In *Proceedings of the Modeling Security Workshop in Association with MODELS 2008*, September 2008.

Tom Mens and Pieter Van Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, March 2006. ISSN 15710661. doi: 10.1016/j.entcs.2005.10.021.

M. Menzel and C. Meinel. A security meta-model for service-oriented architectures. In *Proceedings of the 2009 IEEE International Conference on Services Computing*, SCC '09, pages 251–259, Washington DC, 2009. IEEE Computer Society. ISBN 978-0-7695-3811-2.

M. Menzel and C. Meinel. SecureSOA Modelling Security Requirements for Service-Oriented Architectures. In *2010 IEEE International Conference on Services Computing*, pages 146–153, 2010.

V. Merunka. Critical Assessment of the Role of UML for Information System Development. In *Systems Integration*, pages 445–452, 2003.

J.H. Milam, S.A. Santo, and L.A. Heaton. Concept maps for web-based applications. In *40th Annual Forum of the Association for Instructional Research*, pages 21–24, Cincinnati, May 2000.

Dragan Milicev. On the Semantics of Associations and Association Ends in UML. *IEEE Transactions on Software Engineering*, 33(4):238–251, 2007. ISSN 0098-5589. doi: 10.1109/TSE.2007.37.

MOF. MOF 2 XMI Mapping 2.4.1, August 2011. URL http://www.omg.org/spec/XMI/.

Parastoo Mohagheghi and Vegard Dehlen. Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In Ina Schieferdecker and Alan Hartman, editors, *Model Driven Architecture – Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 432–443. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-69095-5.

Nathalie Moreno, José Raúl Romero, and Antonio Vallecillo. An Overview Of Model-Driven Web Engineering and the Mda. In Gustavo Rossi, Oscar Pastor,

Daniel Schwabe, and Luis Olsina, editors, *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, book part (with own title) 12, pages 353–382. Springer London, 2008. ISBN 978-1-84628-922-4. doi: 10.1007/978-1-84628-923-1_12.

Liping Mu, Terje Gjø saeter, Andreas Prinz, and Merete Skjelten Tveit. Specification of modelling languages in a flexible meta-model architecture. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 302–308, Copenhagen, Denmark, 2010. ACM. ISBN 978-1-4503-0179-4. doi: 10.1145/1842752.1842807.

Pierre-Alain Muller, Frédéric Fondement, and Benoît Baudry. Modeling Modeling. In Bran Schürr, Andy and Selic, editor, *Model Driven Engineering Languages and Systems*, volume 5795 of *Lecture Notes in Computer Science*, book part (with own title) 2, pages 2–16. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-04424-3. doi: 10.1007/978-3-642-04425-0_2.

George C. Necula. Proof-carrying code. In *POPL '97*, POPL '97, pages 106–119, Paris, France, 1997. ACM. ISBN 0-89791-853-3. doi: 10.1145/263699.263712.

George C. Necula and Peter Lee. The design and implementation of a certifying compiler. *SIGPLAN Not.*, 33(5):333–344, May 1998. ISSN 0362-1340. doi: 10.1145/277652.277752.

J.D. Novak and D.B. Gowin. *Learning how to learn*. Cambridge Univ Pr, Cambridge, UK, 1984.

J.D. D Novak and A.J. J Cañas. The Theory Underlying Concept Maps and How to Construct and Use Them. Technical report, Florida Institute for Human and Machine Cognition, Pensacola FL., 2008.

Joseph D Novak and Alberto J Cañas. The origins of the concept mapping tool and the continuing evolution of the tool. *Information Visualization*, 5(3):175–184, 2006.

Martin A. Nowak, Natalia L. Komarova, and Partha Niyogi. Computational and evolutionary aspects of language. *Nature*, 417(6889):611–617, June 2002. ISSN 0028-0836. doi: 10.1038/nature00771.

Florian Noyrit, Sébastien Gérard, François Terrier, and Bran Selic. Consistent Modeling Using Multiple UML Profiles. In Dorina C. Petriu, Nicolas Rouquette, and Ø ystein Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *Lecture Notes in Computer Science*, pages 392–406. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-16144-5. doi: 10.1007/978-3-642-16145-2.

Oasis. Reference Architecture Foundation for Service Oriented Architecture Version 1.0, July 2011. URL http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.html.

Ileana Ober and Andreas Prinz. What do we need metamodels for? In *4th Nordic Workshop on UML and Software Modelling*, pages 8–28, Grimstad, Norway, 2006.

Angela O'Donnell, Donald Dansereau, and Richard Hall. Knowledge Maps as Scaffolds for Cognitive Processing. *Educational Psychology Review*, 14(1):71–86, March 2002. ISSN 1040726X. doi: 10.1023/A:1013132527007.

C.K. Ogden and I.A. Richards. *The meaning of meaning. A study of the influence of language.* Harcourt, Brace, 1923.

A. Okada, S.J.B. Shum, and T. Sherborne. *Knowledge Cartography: software tools and mapping techniques.* Springer-Verlag New York Inc, London, UK, 2008.

T. Olovsson. A structured approach to computer security. Journal article, Chalmers University of Technology, Gothenburg, 1992.

OMG. MetaObject Facility (MOF) 1.4, April 2002.

OMG. MDA Guide Version 1.0.1, June 2003.

OMG. MetaObject Facility (MOF) 2.0 Core specification, January 2006a.

OMG. UML Diagram Interchange, 2006b. URL http://www.omg.org/spec/UMLDI/.

OMG. Diagram Definition RFP-OMG Document 07-09-02, 2007. URL http://www.omg.org/cgi-bin/doc?ad/2007-9-2.

OMG. UML Infrastructure 2.2, 2009a. URL http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/.

OMG. UML Superstructure 2.2, February 2009b. URL http://www.omg.org/spec/UML/2.2/Superstructure/PDF/.

OMG. MetaObject Facility (MOF) 2.4.1 Core specification, 2011a. URL http://www.omg.org/spec/MOF/2.4.1/.

OMG. UML Infrastructure v2.4.1, 2011b. URL http://www.omg.org/spec/UML/2.4.1/.

OMG. UML Superstructure v2.4.1, 2011c. URL http://www.omg.org/spec/UML/2.4.1/.

OMG. OMG Object Constraint Language (OCL) v2.3.1, January 2012a. URL http://www.omg.org/spec/OCL/2.3.1/.

OMG. Service oriented architecture Modeling Language (SoaML) Specification, March 2012b.

Cristian Aurel Opincaru. *Service Oriented Security Architecture applied to Spatial Data Infrastructures*. Journal article, Universität der Bundeswehr München, 2008.

Ing J. F. Overbeek. *Meta Object Facility (MOF) investigation of the state of the art*. Essay (master), University of Twente, 2006.

J. Palomaki. Towards the Foundations of Concept Theory. *Information Modelling and Knowledge Bases V: Principles and Formal Techniques*, 5:139–154, 1994.

Ash Parikh, Anthony Sangha, and Murty Gurajada. Secure your SOA, April 2006. URL http://www.javaworld.com/javaworld/jw-04-2006/jw-0410-webservices.html.

Rachel A. Pottinger and Philip A. Bernstein. Merging models based on given correspondences. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, pages 862–873, Berlin, Germany, 2003. VLDB Endowment. ISBN 0-12-722442-4.

Mohammad Ashiqur Rahaman, Andreas Schaad, and Maarten Rits. Towards secure SOAP message exchange in a SOA. In *Proceedings of the 3rd ACM workshop on Secure web services*, SWS '06, pages 77–84, Alexandria, Virginia, 2006. ACM Press. ISBN 1-59593-546-0.

A. Rensink. Subjects, Models, Languages, Transformations. In J. Bézivin and R. Heckel, editors, *Dagstuhl Seminar Proceedings*, volume 04101 of *Dagstuhl Seminar Proceedings*, Dagstuhl, 2005. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI).

RH Robins. *General linguistics: An introductory survey*. Longman, London, 1971.

Emilio Rodriguez-Priego and Francisco J Garcia-Izquierdo. Securing Code in Services Oriented Architecture. In Luciano Baresi, Piero Fraternali, and Geert-Jan Houben, editors, *Web Engineering*, volume 4607 of *LNCS*, pages 550–555, Como, 2007. Springer. ISBN 978-3-540-73596-0.

Emilio Rodriguez-Priego and Francisco J Garcia-Izquierdo. Integral security model for the exchange of objects in Services Oriented Architecture. In *Proceedings of the 5th International Conference on Web Information Systems and Technologies (WebIST 2009)*, pages 60–65, Lisbon, March 2009. INSTICC Press. ISBN 978-989-8111-81-4.

Emilio Rodriguez-Priego, Francisco García-Izquierdo, and Ángel Rubio. Modeling Issues: a Survival Guide for a Non-expert Modeler. In Dorina Petriu, Nicolas Rouquette, and Ø ystein Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6395 of *Lecture Notes in Computer Science*, book part (with own title) 26, pages 361–375. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-16128-5. doi: 10.1007/978-3-642-16129-2_26.

Emilio Rodriguez-Priego, Francisco J García-Izquierdo, and Ángel Luis Rubio. References-enriched Concept Map: a tool for collecting and comparing disparate definitions appearing in multiple references. *Journal of Information Science*, 2013.

Cristòfol Rovira. DigiDocMap conceptual maps editor and Topic Maps norms, 2005. URL http://eprints.rclis.org/8815/1/DigiDocMap_en__.pdf.

Mehdi Sabbari and Hadiseh Seyyed Alipour. A Security Model and its Strategies for Web Services. *International Journal of Computer Applications*, 36(10):24–31, December 2011.

M.Q. Saleem, J. Jaafar, and M.F. Hassan. Model Driven Security framework for definition of security requirements for SOA based applications. In *International Conference on Computer Applications & Industrial Electronics (ICCAIE 2010)*, pages 266–270, Kuala Lumpur, 2010.

Diana M. Sánchez, José María Cavero, and Esperanza Marcos. The concepts of model in information systems engineering: a proposal for an ontology of models. *The Knowledge Engineering Review*, 24(Special Issue 01):5–21, 2009. doi: 10.1017/S0269888909000150.

F. Satoh, Y. Nakamura, N.K. Mukhi, M. Tatsubori, and K. Ono. Model-Driven Approach for End-to-End SOA Security Configurations. In Nikola Milanovic, editor, *Non-Functional Properties in Service Oriented Architecture: Requirements, Models and Methods*, pages 268–298. IGI Global, 2010. ISBN 9781605667942.

Pekka Savolainen, Eila Niemela, and Reijo Savola. A Taxonomy of Information Security for Service-Centric Systems. In *Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on*, pages 5–12, Lubeck, Germany, August 2007. IEEE.

Robert M. Schwartz and Taffy E. Raphael. Concept of Definition: A Key to Improving Students' Vocabulary. *The Reading Teacher*, 39(2):198—-205, 1985. ISSN 00340561. doi: 10.2307/20199044.

E. Seidewitz. What Models Mean. *IEEE Software*, 20(5):26–32, September 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231147.

B. Selic. What's new in UML 2.0, 2005. URL `ftp://129.35.224.112/software/rational/web/whitepapers/intro_uml2.pdf`.

Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. *SIGOPS Oper. Syst. Rev.*, 39(5):1–16, December 2005. doi: 10.1145/1095810.1095812.

Lijun Shan and Hong Zhu. A Formal Descriptive Semantics of UML. In Shaoying Liu, Tom Maibaum, and Keijiro Araki, editors, *Formal Methods and Software Engineering*, volume 5256 of *LNCS*, pages 375–396. Springer, 2008. ISBN 978-3-540-88193-3. doi: 10.1007/978-3-540-88194-0_23.

John Sowa. Ontology, Metadata, and Semiotics. In *Conceptual Structures: Logical, Linguistic, and Computational Issues*, volume 1867 of *Lecture Notes in Computer Science*, book part (with own title) 5, pages 55–81–81. Springer Berlin / Heidelberg, 2000. ISBN 978-3-540-67859-5. doi: 10.1007/10722280_5.

John Sowa and Arun Majumdar. Analogical Reasoning. In *Conceptual Structures for Knowledge Creation and Communication*, pages 16–36. Springer Berlin Heidelberg, 2003.

Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer Wien, 1973.

Peter Frederick Strawson. On referring. *Mind*, 59(235):320–344, 1950.

Garret Swart, Benjamin Aziz, Simon N. Foley, and John Herbert. Trading off security in a service oriented architecture. In Duminda Jajodia, Sushil and Wijesekera, editor, *Data and Applications Security XIX*, volume 3654 of *Lecture Notes in Computer Science*, pages 924–924. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-28138-2.

Chris Swoyer and Francesco Orilia. Properties. In Edward N Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 201 edition, 2011. URL `http://plato.stanford.edu/archives/win2011/entries/properties/`.

JA Sykes. Negotiating early reuse of components: a model-based analysis. In *UML and the unified process*, pages 66–69. IGI Publishing, 2003.

Tanit Talbi, Bertrand Meyer, and Emmanuel Stapf. A Metric Framework for Object-Oriented Development. In *Technology of Object-Oriented Languages, International Conference on*, pages 164–172. IEEE Computer Society, 2001. ISBN 0-7695-1251-8. doi: 10.1109/TOOLS.2001.941670.

Kezhe Tang, Shiping Chen, D. Levy, J. Zic, and B. Yan. A Performance Evaluation of Web Services Security. In *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, pages 67–74. IEEE Computer Society, 2006.

Marie Noelle Terrasse, Marinette Savonnet, Eric Leclercq, Thierry Grison, and George Becker. Do we need metamodels AND ontologies for engineering platforms? In *Proceedings of the 2006 international workshop on Global integrated model management*, pages 21–28, Shanghai, China, 2006. ACM. ISBN 1-59593-410-3. doi: 10.1145/1138304.1138310.

The Open Group. Service-Oriented Architecture Ontology, October 2010.

The Open Group. SOA Reference Architecture, December 2011.

TheOpenGroup, OMG, and OASIS. Navigating the SOA Open Standards Landscape Around Architecture, 2009.

D. Thomas. MDA:Revenge of the modelers or UML utopia? *Software, IEEE*, 21 (3):15–17, 2004.

Dave Thomas. What's in an Object. *Byte*, 14(3):231–240, 1989.

Marco Torchiano, Federico Tomassetti, Filippo Ricca, Alessandro Tiso, and Gianna Reggio. Preliminary Findings from a Survey on the MD State of the Practice. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, ESEM '11, pages 372–375. IEEE Computer Society, 2011. ISBN 978-0-7695-4604-9.

C.R. Twardy. Argument maps improve critical thinking. *Teaching Philosophy*, 27(2):95—-116, 2003.

Stephen Ullmann. *Semantics An Introduction to the Science of Meaning*. Basil Blackwell, Oxford, 1962.

Dániel Varró and András Pataricza. A Unifying Semantic Framework for Multilevel Metamodeling. Technical report, Budapest University of Technology and Economics, October 2001.

Eelco Visser. WebDSL: A Case Study in Domain-Specific Language Engineering. In Ralf Lämmel, Joost Visser, and João Saraiva, editors, *Generative and Transformational Techniques in Software Engineering II*, volume 5235 of *Lecture Notes in Computer Science*, journal article 7, pages 291–373. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-88642-6. doi: 10.1007/978-3-540-88643-3_7.

H. Wada and J. Suzuki. A Model-Driven Development framework for Non-functional aspects in service oriented architecture. *International Journal of Web Services Research*, 5(4):1–31, 2008.

Hiroshi Wada, Junichi Suzuki, and Katsuya Oba. A Service-Oriented Design Framework for Secure Network Applications. In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, volume 01 of *COMPSAC '06*, pages 359–368, Washington DC, 2006. IEEE Computer Society. ISBN 0-7695-2655-1.

Gerd Wagner, Carlos Viegas Damasio, and Grigoris Antoniou. Towards a general web rule language. *Int. J. Web Eng. Technol.*, 2(2/3):181–206, 2005. doi: 10.1504/IJWET.2005.008483.

Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, Upper Saddle River, NJ, USA, March 2005. ISBN 0131488740.

Ingo Weisemöller and Andy Schürr. A Comparison of Standard Compliant Ways to Define Domain Specific Languages. In Holger Giese, editor, *Models in Software Engineering*, volume 5002 of *LNCS*, pages 47–58. Springer, 2008. ISBN 978-3-540-69069-6. doi: 10.1007/978-3-540-69073-3_6.

Yang Xia and Martin Glinz. Rigorous EBNF-based definition for a graphic modeling language. In *Software Engineering Conference, 2003. Tenth Asia-Pacific*, pages 186–196, 2003. doi: 10.1109/APSEC.2003.1254371.

Zhichen Xu, Barton P. Miller, and Thomas Reps. Safety checking of machine code. In *SIGPLAN Not.*, volume 35 of *PLDI '00*, pages 70–82, Vancouver, British Columbia, Canada, May 2000. ACM. ISBN 1-58113-199-2. doi: 10.1145/349299.349313.

K. Yskout, B. De Win, and W. Joosen. Transforming security audit requirements into a software architecture. In *Proceedings of the Workshop on Modeling Security (MODSEC08)*, volume 413, pages 1–10, 2008.

Edward N. Zalta, Uri Nodelman, Colin Allen, and John Perry. Concepts (Stanford Encyclopedia of Philosophy/Spring 2009 Edition), 2011. URL http://plato.stanford.edu/archives/spr2009/entries/concepts/.